|  |  |  |
| --- | --- | --- |
|  | UNIVERSITATEA DIN CRAIOVA  FACULTATEA DE AUTOMATICĂ, CALCULATOARE ȘI ELECTRONICĂ  DEPARTAMENTUL DE CALCULATOARE ȘI TEHNOLOGIA INFORMAȚIEI |  |

PROIECT DE DIPLOMĂ

ANDREI IRIMIA

COORDONATOR ȘTIINȚIFIC

Șef lucr.dr.ing. DUMITRAȘCU EUGEN

IULIE 2025

CRAIOVA

|  |  |  |
| --- | --- | --- |
|  | UNIVERSITATEA DIN CRAIOVA  FACULTATEA DE AUTOMATICĂ, CALCULATOARE ȘI ELECTRONICĂ  DEPARTAMENTUL DE CALCULATOARE ȘI TEHNOLOGIA INFORMAȚIEI |  |

REMOTE CONTROLLED CAR

ANDREI IRIMIA

COORDONATOR ȘTIINȚIFIC

Șef lucr.dr.ing. DUMITRAȘCU EUGEN

IULIE 2025

CRAIOVA

*„Învățătura este o comoară care își urmează stăpânul pretutindeni.”*

Proverb popular

**DECLARAȚIE DE ORIGINALITATE**

Subsemnatul ANDREI IRIMIA, student la specializarea CALCULATOARE SI TEHNOLOGIA INFORMAȚIEI ENGLEZA, din cadrul Facultății de Automatică, Calculatoare și Electronică a Universității din Craiova, certific prin prezenta că am luat la cunoştinţă de cele prezentate mai jos şi că îmi asum, în acest context, originalitatea proiectului meu de licenţă:

* cu titlul REMOTE CONTROLLED CAR,
* coordonată de Șef lucr.dr.ing. DUMITRAȘCU EUGEN,
* prezentată în sesiunea IULIE 2025.

La elaborarea proiectului de licenţă, se consideră plagiat una dintre următoarele acţiuni:

* reproducerea exactă a cuvintelor unui alt autor, dintr-o altă lucrare, în limba română sau prin traducere dintr-o altă limbă, dacă se omit ghilimele şi referinţa precisă,
* redarea cu alte cuvinte, reformularea prin cuvinte proprii sau rezumarea ideilor din alte lucrări, dacă nu se indică sursa bibliografică,
* prezentarea unor date experimentale obţinute sau a unor aplicaţii realizate de alţi autori fără menţionarea corectă a acestor surse,
* însuşirea totală sau parţială a unei lucrări în care regulile de mai sus sunt respectate, dar care are alt autor.

Pentru evitarea acestor situaţii neplăcute se recomandă:

* plasarea între ghilimele a citatelor directe şi indicarea referinţei într-o listă corespunzătoare la sfărşitul lucrării,
* indicarea în text a reformulării unei idei, opinii sau teorii şi corespunzător în lista de referinţe a sursei originale de la care s-a făcut preluarea,
* precizarea sursei de la care s-au preluat date experimentale, descrieri tehnice, figuri, imagini, statistici, tabele et caetera,
* precizarea referinţelor poate fi omisă dacă se folosesc informaţii sau teorii arhicunoscute, a căror paternitate este unanim cunoscută și acceptată.

Data, Semnătura candidatului,

|  |  |  |
| --- | --- | --- |
|  | UNIVERSITATEA DIN CRAIOVA  Facultatea de Automatică, Calculatoare şi Electronică  Departamentul de Calculatoare și Tehnologia Informației | Aprobat la data de  …………………  Şef de departament,  Prof. dr. ing.  Nicolae Enescu/  Comin IONETE/  Dorian COJOCARU |

**PROIECTUL DE DIPLOMĂ**

|  |  |
| --- | --- |
| Numele și prenumele studentului/-ei: | IRIMIA ANDREI |
| Enunțul temei: | REMOTE CONTROLLED CAR |
| Datele de pornire: | Dezvoltarea unui sistem hardware-software care utilizeaza diferite microcontrolere si senzori pentru comandarea unei platforme mobile cu ajutorul gesturilor mainii. |
| Conținutul proiectului: | Aplicatia hardware-software contine urmatoarele:   * Detectarea si masurarea nivelurilor de inclinatie de mainii pe toate axele. * Transmiterea informatiei wireless de la telecomanda la masina. * Prelucrarea informatiei si ajustarea directiei masinii. * Detectia de obstacole. |
| Material grafic obligatoriu: | Scheme, figuri si diagrame |
| Consultații: | Lunare |
| Conducătorul științific  (titlul, nume și prenume, semnătura): | Șef lucr.dr.ing. DUMITRAȘCU EUGEN |
| Data eliberării temei: |  |
| Termenul estimat de predare a proiectului: |  |
| Data predării proiectului de către student și semnătura acestuia: |  |

|  |  |  |
| --- | --- | --- |
|  | UNIVERSITATEA DIN CRAIOVA  Facultatea de Automatică, Calculatoare şi Electronică  Departamentul de Calculatoare și Tehnologia Informației |  |

**REFERATUL CONDUCĂTORULUI ȘTIINȚIFIC**

|  |  |
| --- | --- |
| Numele și prenumele candidatului/-ei: |  |
| Specializarea: | [*Denumirea oficială a specializării absolvite de candidat*] |
| Titlul proiectului: | [*Titlul lucrării*] |
| Locația în care s-a realizat practica de documentare (se bifează una sau mai multe din opțiunile din dreapta): | În facultate □ |
| În producție □ |
| În cercetare □ |
| Altă locație: [*se detaliază*] |

În urma analizei lucrării candidatului au fost constatate următoarele:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nivelul documentării | | Insuficient  □ | Satisfăcător □ | Bine  □ | Foarte bine  □ |
| Tipul proiectului | | Cercetare  □ | Proiectare  □ | Realizare practică □ | Altul  [*se detaliază*] |
| Aparatul matematic utilizat | | Simplu  □ | Mediu  □ | Complex □ | Absent  □ |
| Utilitate | | Contract de cercetare □ | Cercetare internă □ | Utilare  □ | Altul  [*se detaliază*] |
| Redactarea lucrării | | Insuficient  □ | Satisfăcător □ | Bine  □ | Foarte bine  □ |
| Partea grafică, desene | | Insuficientă  □ | Satisfăcătoare □ | Bună  □ | Foarte bună  □ |
| Realizarea practică | Contribuția autorului | Insuficientă  □ | Satisfăcătoare □ | Mare  □ | Foarte mare  □ |
| Complexitatea  temei | Simplă  □ | Medie  □ | Mare  □ | Complexă  □ |
| Analiza cerințelor | Insuficient  □ | Satisfăcător □ | Bine  □ | Foarte bine  □ |
| Arhitectura | Simplă  □ | Medie  □ | Mare  □ | Complexă  □ |
| Întocmirea specificațiilor funcționale | Insuficientă  □ | Satisfăcătoare □ | Bună  □ | Foarte bună  □ |
| Implementarea | Insuficientă  □ | Satisfăcătoare □ | Bună  □ | Foarte bună  □ |
| Testarea | Insuficientă  □ | Satisfăcătoare □ | Bună  □ | Foarte bună  □ |
| Funcționarea | Da  □ | Parțială  □ | Nu  □ | |
| Rezultate experimentale | | Experiment propriu  □ | | Preluare din bibliografie  □ | |
| Bibliografie | | Cărți | Reviste | Articole | Referințe web |
| Comentarii  și  observații | |  | | | |

În concluzie, se propune:

|  |  |
| --- | --- |
| ADMITEREA PROIECTULUI  □ | RESPINGEREA PROIECTULUI  □ |

Data, Semnătura conducătorului științific,

**PROJECT SUMMARY**

Microcontrollers are essential components in everyday life, being integrated into numerous smart and electronic devices.

This project aims to present a hardware-software system for controlling a moving platform using hand gestures.

Various hardware technologies were used in the implementation of this project, such as the ESP32 and Arduino UNO development boards, the MPU-9250 sensor (gyroscole, magnetometer and accelerometer), the HC-SR04 ultrasonic sensor and L293D motor driver. Additionally, communication between modules was achieved using protocols such as UART and I2C. Communication between the ESP32 modules was established via Wi-Fi, using the TCP/IP protocol.

Some of the unique aspects of this project include:

* Gesture control: Hand movements are translated into commands for the vehicle.
* Obstacle detection: The vehicle detects direct obstacles and disables forward movement when necessary.
* Modular architecture: This software is modular, allowing for easy expansion and the addition of new functionalities.

Through the development of this project, I have gained valuable knowledge of integrating and managing various hardware modules in a practical context. I have learned how to connect and configure components such as sensors, development boards, and communication modules, ensuring their coordinated operation in an integrated system.

***Termenii cheie***: hardware, communication, vehicle, gesture control.

**TABLE OF CONTENTS**

[1 Introducere 1](#_Toc309895962)

[1.1 Scopul 1](#_Toc309895963)

[1.2 Motivația 1](#_Toc309895964)

[2 Convenții de redactare 2](#_Toc309895965)

[2.1 Cerințe generale 2](#_Toc309895966)

[2.2 Structura documentului 2](#_Toc309895967)

[2.3 Dimensiunile lucrării 3](#_Toc309895968)

[2.4 Elemente de tehnoredactare 3](#_Toc309895969)

[2.5 Formulele matematice 4](#_Toc309895970)

[2.6 Ilustrațiile 4](#_Toc309895971)

[2.6.1 Figurile 4](#_Toc309895972)

[2.6.2 Tabelele 4](#_Toc309895973)

[2.6.3 Legenda (unei figuri/tabele) 6](#_Toc309895974)

[3 Termeni de utilizare 7](#_Toc309895975)

[3.1 Autorii 7](#_Toc309895976)

[3.2 Licența de utilizare 7](#_Toc309895977)

[4 Concluzii 8](#_Toc309895978)

[5 Bibliografie 9](#_Toc309895979)

[6 Referințe web 10](#_Toc309895980)

[A. Codul sursă 11](#_Toc309895981)

[B. Site-ul web al proiectului 12](#_Toc309895982)

[C. CD / DVD 13](#_Toc309895983)

[Index 14](#_Toc309895984)

**LISTA FIGURILOR**

[Figura 1. Selectarea prin click dreapta a opțiunii „Update field” 5](#_Toc309893908)

[Figura 2. Actualizarea întregului tabel 5](#_Toc309893909)

**LISTA TABELELOR**

[Tabelul 1. Nume de utilizatori și valorile rezumat ale parolelor acestora 5](#_Toc309893145)

# INTRODUCTION

Microcontrollers have become a fundamental part of modern embedded systems, enabling the development of smart, interactive, and autonomous devices. This project presents a remote-controlled vehicle operated via a wireless glove interface, utilizing gesture recognition and real-time obstacle detection. The system leverages the capabilities of ESP32 and Arduino UNO microcontrollers, integrating wireless communication, sensor fusion, and safety logic to create an intuitive and robust control platform.

The glove serves as the central and most innovative component of the project’s control system. Wearing the glove, the driver instinctively manipulates the vehicle simply by moving his hand. The glove contains an inertial measurement unit that continuously tracks the orientation and movement of the hand. These motions of the hands – lean forward, lean backward or lean sideways – are converted into driving commands (forward, backward, left, right, stop). An ESP32 microcontroller processes the sensor data in real time and transmits the corresponding commands wirelessly to the vehicle using a fast and efficient communication protocol. The glove thus exemplifies how wearable technology can create a seamless and enjoyable human-machine interface.

## Scope

The scope of this project encompasses several key areas in the field of human-machine interaction, embedded systems, and wireless control using wearable technology. The main focus here is on the creation of an intuitive and modular remote-controlled vehicle system, which is controlled using a gesture-based glove interface. It entails motion sensing, wireless communication, and real-time control that work to provide an integrated and interesting experience for the users. This rides on the developing trend of intuitiveness of wearables for control applications, as witnessed in works such as Lin et al., 2019[[LIN20](#LIN20)], where gestures are applied in robotic arm manipulation.

While the project covers a broad range of topics, it does not attempt to address all aspects of gesture recognition, wireless communication, or robotics. For example, advanced machine learning techniques for gesture classification are beyond the scope of this work.

The limitations of the chosen technologies are acknowledged. Gesture recognition is based on simple thresholding rather than advanced pattern recognition, and obstacle detection relies on a single ultrasonic sensor, which may not provide full environmental awareness.

In conclusion, this project aims to provide a practical demonstration of gesture-based vehicle control using wearable technology and embedded microcontrollers. It offers a comprehensive overview of the key techniques and challenges in creating modern, user-friendly human-machine interfaces, and serves as a foundation for further research and development in this rapidly evolving field.

## Motivation

The motivation behind this project is to learn new, more natural methods through which human beings can interact with and instruct machines. While the existing implementation is aimed at small remotely controlled machines, the technology and concepts involved are scalable and adaptable to much larger and more complex machines.

I chose this project because I am passionate about the field of embedded systems and human-machine interaction. I find it fascinating how microcontrollers, sensors, and wireless technologies can be combined to create innovative and practical solutions that bridge the gap between the digital and physical worlds. Working on gesture-based control and real-time communication allowed me to explore both the technical and creative aspects of engineering. This field is constantly evolving, with endless opportunities for learning, experimentation and real-world impact – whether in robotics, automation, or everyday smart devices.

## Project Vision

The vision of this project is to demonstrate a new, more natural way for people to interact with machines, moving beyond traditional remote controls and interfaces. By focusing on the seamless translation of human intention into machine action, the project aims to make technology more approachable, responsive and adaptable to the needs of its users.

At its heart, the project is about creating a system where the boundaries between human commands and machine responses are minimized. The remote-controllable vehicle is designed to respond to user commands in a way that feels instantaneous and intuitive, allowing one to enjoy a more fulfilling and interactive experience. Not only does this enhance usability, but it also offers new paths for how people of all ages and backgrounds can engage with technology.

A defining feature of the system is the modularity of design. Each component – user interface, vehicle's control unit, and the motor driver – operates independently but in harmony with the others. The modularity ensures that it is easy to reconfigure the system for various applications, scale it when necessary, or redefine it for new purposes.

Furthermore, the project highlights the importance of real-time feedback and responsiveness in modern systems. The vehicle’s ability to react instantly to user commands and environmental changes demonstrates the potential for creating interactive experiences that are both reliable and enjoyable.

In summary, the vision of this project is to pave the way forward for more intuitive, flexible and human-centered interactions with machines.

# ANALYSIS AND DESIGN OF THE SYSTEM

## Project Structure

In the following chapters, the overall structure of the system will be presented, along with the software and hardware architectures and schematics.

## List of Hardware Components

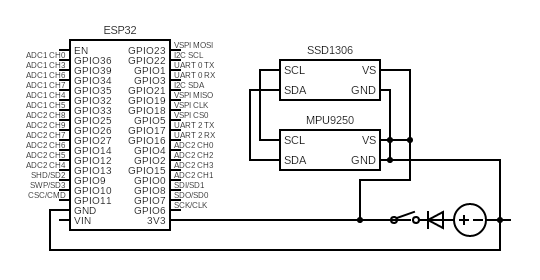
This chapter provides a detailed inventory of all hardware parts used in the project. Each component is listed with the number of pieces used in the project and a short description of its usage.

|  |  |  |
| --- | --- | --- |
| Amount | Component | Usage |
| 2 | ESP32 | Glove and vehicle control |
| 1 | MPU9250 | Tilt information of the glove |
| 1 | SSD1306 OLED | Display tilt inclination on the glove |
| 7 | 18650 Battery | Power supply for the whole system |
| 2 | SPDT Switch | Power supply switch |
| 1 | 1N4007 Diode | Reverse polarity protection |
| 1 | HC-SR04 | Ultrasonic sensor for obstacle detection |
| 1 | Arduino UNO | Motor driver control |
| 1 | L293D | Motor driver for motor control |
| 2 | 2200 uF Capacitor | Noise filtering on the motor driver |
| 1 | Powerbank 18650 4P 5V / 3V DC | Module for steady 5V output to the vechicle circuit |
| 4 | Geared Motor DC | Facilitates movement of the vehicle |

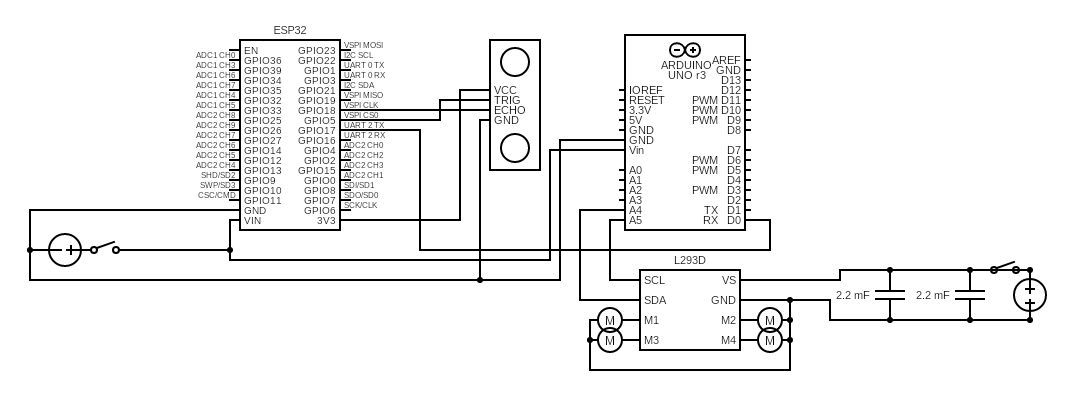
## Schmatic Diagrams

This chapter presents the detailed electrical schematics of the system, illustrating the physical connections between components, sensors and actuators. The diagrams provide a clear visual representation of the hardware setup, enabling a better understanding of the system architecture.

The glove (controller) schematic shows the connections between the gyroscope, OLED display and microcontroller.



The car schematic illustrates how the main components of the vehicle are connected. Key components are: ESP32, ultrasonic sensor, Arduino UNO, Motor Driver.



## System Requirements

**ID:** FR1

**Feature:** Hand Movement Input

**Contents:** The function *Remote\_Glove* shall <<be able to capture hand movement information by the means of a gyroscope sensor and compute raw data into percentage value for tilt.>>.

**ID:** FR2

**Feature:** Display OLED Data

**Contents:** The function *Remote\_Glove* shall <<establish the I2C communication line with the physical OLED display and transfer information about gyroscope processes data and Wi-Fi communication status.>>.

**ID:** FR3

**Feature:** Transfer Processes Gyroscope Data

**Contents:** The functions *Remote\_Glove* and *Remote\_Car* shall <<be able to establish a Wi-Fi communication with each other for transfer of gyroscope-processed information.>>.

**ID:** FR4

**Feature:** Compute Drive Command

**Contents:** The function *Remote\_Car* shall <<compute gyroscope percentage and ultrasonic sensor data and compose a driving command depending on hand gestures. The function should not allow the direction FORWARD to take effect when an obstacle is present less than 20 cm of the car.>>.

**ID:** FR5

**Feature:** Send UART Command

**Contents:** The functions *Remote\_Car* and *Motor\_Driver* shall <<establish the UART communication with each other and transfer the final driving command.>>.

**ID:** FR6

**Feature:** Process Driving Command

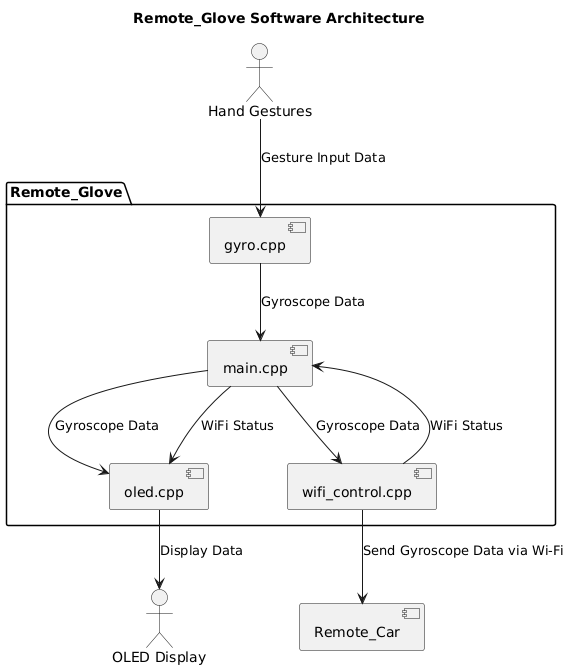
**Contents:** The function *Motor\_Driver* shall <<transform the driving command into corresponding L293D data and transmit it to the driver.>>.

### Software architecture

This chapter presents the high-level design of the software components involved in the project. It describes the modular structure, the interaction between the main functional blocks, and the data flow between software modules and external hardware. The focus is outlining clear responsibilities for each module, enabling maintainability, scalability, and efficient communication within the system.

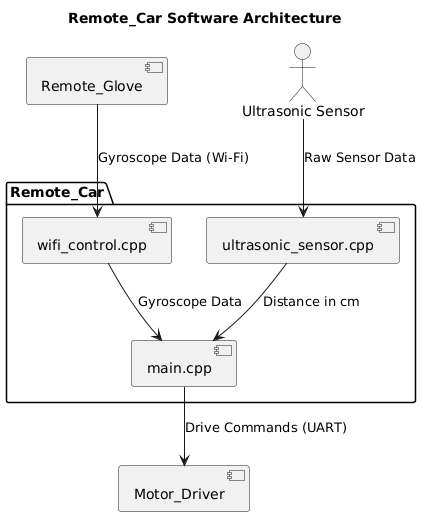
The *Remote\_Glove* software block is responsible for capturing and processing user hand gestures and transmitting the corresponding control signals wirelessly. It consists of several submodules:

* gyro.cpp: Reads raw data from the gyroscope sensor, processes it to compute roll and pitch angles as percentage values, and sends this data to the main control module.
* oled.cpp: Receives processes sensor data and Wi-Fi connection status from the main module and displays them on the OLED screen, providing real-time feedback to the user.
* wifi\_control.cpp: Manages wireless communication by transmitting gyroscope data to the *Remote\_Car* software block and receiving connection updates, which are then forwarded to the main module for display.
* main.cpp: Manages, requests and sends information to the other modules in this software block.



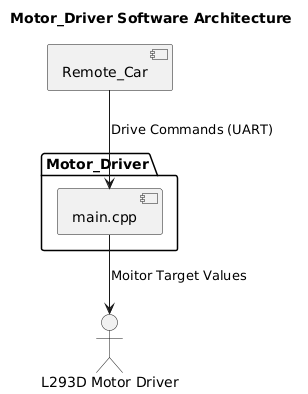
The *Remote\_Car* software block handles receiving control signals from the *Remote\_Glove* and processes sensor data for safe and accurate vehicle operation. It consists of the following submodules:

* ultrasonic\_sensor.cpp: Interfaces with the ultrasonic sensor to measure distance to obstacles in centimeters, providing real-time environmental awareness to avoid collisions.
* wifi\_control.cpp: Receives gyroscope data transmitted wirelessly from the *Remote\_Glove* software block and passes this information to the main control unit.
* main.cpp: Acts as the central processing unit of the Remote\_Car software block. It integrates data from the gyroscope sensor and ultrasonic sensor, processes this information to determine appropriate driving commands and sends these commands to the *Motor\_Driver* software block via UART for execution.



The Motor\_Driver software block is responsible for translating high-level drive commands into precise motor control signals for the external L293D Motor Driver hardware.

By isolating motor control within this dedicated module, the system ensures a clear separation of concerns, enabling easier maintenance and potential upgrades to the motor control logic without affecting higher-level modules.



## Product Functions

The system offers the following functionalities:

* Provides a natural hand-gesture control system, with the capacity to navigate a car through the movement of one's hand while wearing a special glove.
* Enables real-time wireless communication between the glove and the vehicle, ensuring an immediate response to user commands without the need for physical buttons or joysticks.
* Supports multiple movement commands, including forward, backward, left, right, and stop, all triggered by natural hand gestures.
* Integrates obstacle detection functionality, automatically preventing the vehicle from moving forward if an object is detected within a critical distance.
* Offers expandability, allowing the system to be versatile for different types of vehicles or upgradeable with new features if needed.

## User Characteristics

The main user will be the individual operating the gesture-controlled vehicle system. This user can be a student, hobbyist, or anyone interested in exploring modern human-machine interaction. The user should be able to control the vehicle intuitively using hand gestures and observe the vehicle’s responses in real time.

# TOOLS AND TECHNOLOGIES

## Microcontroller Platforms

### Microcontroller Platforms

Integrated circuits called microcontrollers are at the heart of a wide range of electronic devices, from industrial machinery to consumer goods. In order to carry out particular control tasks in a dependable and power-efficient manner, they integrate a processor, memory, and input/output peripherals onto a single chip. In embedded systems, microcontrollers are widely used to process sensor data, carry out preprogrammed actions, and manage actuators or other hardware elements.[[BAR06](#BAR06)][[MAZ13](#MAZ13)]

### ESP32 Platform

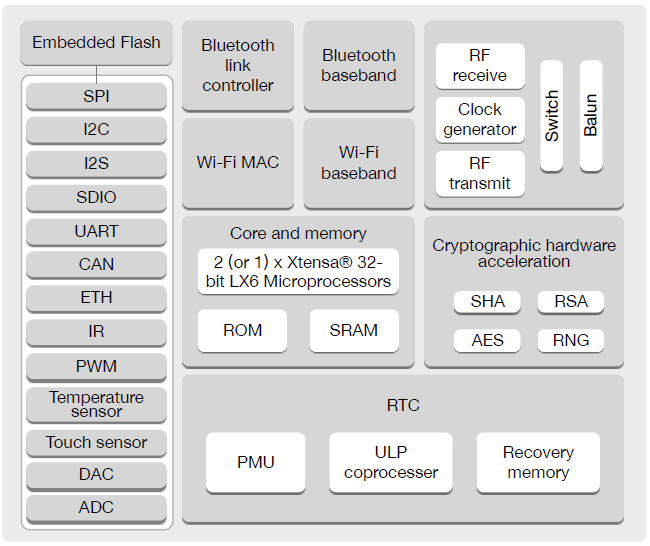
ESP32 is an integrated microcontroller platform created by Espressif Systems. The microcontroller device comes built-in with Wi-Fi and Bluetooth, allowing fast connectivity and communication with other ESP32 devices due to its fast dual-core processor.[[ESPD23](#ESPD23)]

One of the most important features for the ESP32 development board is its integrated wireless module, which facilitates communication over Wi-Fi or Bluetooth without the need for additional ICs. Projects that require remote control, data transmission, or real-time monitors take full advantage of those features. The ESP32 comes packed with a rich set of peripherals, including analog and digital interfaces, timers, and communication protocols, enabling the developer to connect to a wide range of sensors and actuators.[[ESPT23](#ESPT23)]

The ESP32 development module includes a dual-core Tensilica Xtensa LX6 32-bit processor, which is capable of running at frequencies up to 240 MHz. This enables fast and efficent execution of real-time tasks and high-level network communication simultaneously. The Wi-Fi module can be configured either in station or access point modes and enables direct peer-to-peer communication or integration with existing networks. The Bluetooth module allows both classic and BLE modes, useful for low-latency data exchange with wearables or mobile devices. The ESP32 includes a rich set of peripherals and GPIOs:

* 18 channels of 12-bit ADC
* 2 channels of 8-bit DAC
* UART, SPI, I2C, PWM, and CAN

This extensive I/O flexibility allows the ESP32 to directly interface with sensors, displays, and actuators, minimizing the need for external ICs.[[ESPD23](#ESPD23)]



In this project, the ESP32 was chosen for both the glove and the vehicle controller due to its wireless capabilities, processing power, and ease of integration with various sensors. Its open-source development environment and strong community support further simplify the process of prototyping and deploying embedded systems.

### Arduino UNO Platform

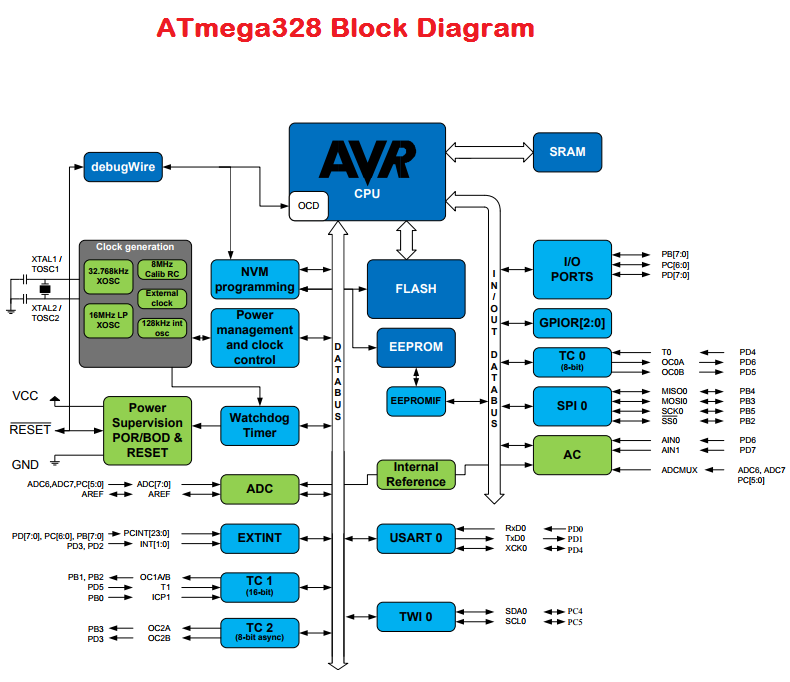
Arduino UNO is a popular development board based on the ATmega328P microcontroller. It offers a reliable and easy-to-use platform for control tasks, making it an excellent choice for dedicated control modules such as motor drivers.

The development board is equipped with an 8-bit AVR RISC processor known as ATmega328P running at a frequency of 16 MHz. The processor is less powerfull than the ESP32 processor, but it is well-suited for controlling single modules such as the L293D motor driver due to its simplicity and predictable timing. The Arduino UNO board has 32 KB of flash memory for program storage, 2KB of SRAM, and 1KB of EEPROM for non-volatile data storage.

The board provides a wide variety of interfaces and control features, such as:

* 14 digital input/output pins, of which 6 can be used for PWM output.
* 6 analog input pins, which can be used for sensor readings or feedback.
* Communication protocols such as UART, SPI, and I2C, allowing interfacing with other modules if needed.[[ARDU23](#ARDU23)]

The processor present on the board is based on the modified Harvard architecture, allowing simultaneous access to instructions and data, which improves performance for real-time embedded tasks.[[MICR15](#MICR15)]



[[ENPR17](#ENPR17)]

In this system, the Arduino UNO acts as the motor driver controller, receiving drive commands via UART from the ESP32-based remote car controller. It processes these commands to generate appropriate PWM signals and digital outputs that drive the L293D motor driver module, which in turn powers the DC geared motors.

### Comparison and Integration

For this project, both ESP32 and Arduino UNO are used, each with its own capabilities. The ESP32 performs better at handling multiple sensors and real-time data; also, the wireless capabilities make it ideal for projects that require remote control. On the other hand, the simplicity of the Arduino UNO makes it ideal for easy tasks such as handling the attached motor driver.

The integration of these two platforms allows the system to perform at its peak. The ESP32 handles the complex tasks of gesture recognition and wireless data transmission on the glove, and sensor readings on the vehicle, while the Arduino UNO focuses on executing precise motor commands. Communication between the ESP32 and the Arduino UNO is achieved via a serial connection.

The division of responsibilities makes the whole system optimized, opening the gates for future expansion. Each platform can be developed and tested independently, making the system more flexible and easy to mentain.

## Sensors and Actuators

### Overview

Sensors and actuators are critical components in modern electronic systems that facilitate the interaction between humans and machines.

Sensors are electronic devices that detect and measure physical properties of the environment. They convert such physical stimuli as temperature, light, motion, pressure, or humidity into electrical signals that can be interpreted by electronic systems. Sensors gather real-time data, monitor conditions, and enable systems to respond to changes in their surroundings.

Actuators can be considered the complete opposite of sensors; they control signals from the electronic systems and convert them into physical action. Actuators can move or control a mechanism or system, such as opening valves or rotating motors.

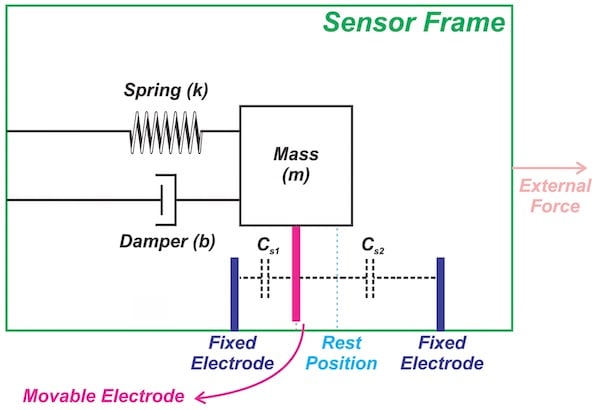
Together, sensors and actuators form the backbone of most electronic or embedded systems, from simple household appliances to complex industrial machinery. Their integration enables the creation of responsive, adaptive, and intelligent systems that can sense, decide, and act in real time.

### IMU MPU9250 Motion Sensor

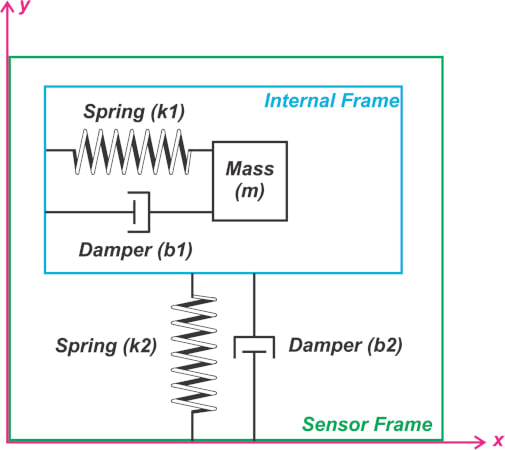
The MPU9250 is an inertial measurement unit developed by InvenSense that combines three sensors into a single IC:

* A 3-axis gyroscope.
* A 3-axis accelerometer.
* A 3-axis magnetometer. [[INVE13](#INVE13)]

The 3-axis capacitive accelerometer measures linear acceleration in the X, Y, and Z directions, allowing the detection of tilting, vibration, or movements such as lifting or shaking the device. Capacitive accelerometers measure the proof mass displacement by taking into consideration the capacitance between a movable electrode and a fixed one. The mobile electrode is moved whenever the sensor is shaken or displaced.[[ALLA21](#ALLA21)]



The 3-axis vibratory MEMS gyroscope is designed to measure angular velocity across the X, Y, and Z axes. In simple terms, it measures the rate of rotation of the sensor by a series of intricate mathematical formulas.[[ALLG22](#ALLG22)]



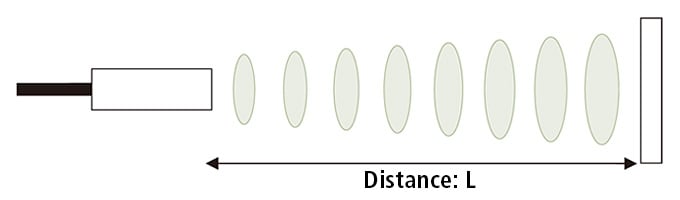
The 3-axis magnetometer present in the sensor measures the strength and direction of magnetic fields along the three orthogonal axes. The magnetometer used in the MPU9250 focuses on Hall-effect technology by sensing the voltage on an electrical conductor when applying a magnetic field.[[ALLM21](#ALLM21)]

In the presented project, only the accelerometer part of the MPU9250 is used to calculate roll and pitch angles using trigonometric formulas.

### HC-SR04 Ultrasonic Sensor

The HC-SR04 ultrasonic sensor is used for distance measurement in embedded systems. It operates by using a single ultrasonic element for both transmission and reception. The distance is measured by emitting a 40 kHz sound that reflets back from the nearest object. The time that it takes for the sound to travel back and forth is taken into consideration by the sensor to calculate the distance to the object using the following formula:

The HC-SR04 contains only one ultrasonic element that switches between sending and receiving signals.[[KEYE20](#KEYE20)]

14

Although it is a fantastic tool to measure distance in embedded projects, the sensor comes with its own disadvantages, such as:

* Limited Range.
* Can be distrubed by other sound-emitting sources.
* Can have problems detecting objects of high density due to sound properties.[[ROBO23](#ROBO23)]

### L293D Motor Driver Arduino Shield

The L293D Arduino Shield is designed to fit directly on top of an Arduino UNO development board. It provides a way to control multiple motors using the L293D H-bridge driver ICs. The shield can be used to drive up to 4 bi-directional DC motors or up to 2 stepper motors.

The L293D chip is a quadruple half-H driver designed to allow bidirectional current flow through inductive loads such as DC motors. It was originally developed by Texas Instruments.[[TEXA19](#TEXA19)]

The driver receives logic signals from the microcontroller to determine the direction and enable/disable each motor. It uses an H-bridge configuration internally to switch the higher motor voltage and current, allowing it to move forward, backward, or stop.

In this project the L293D motor driver is controlled by the attached Arduino UNO, its main function being to control the speed of the motors. The shield has its own ~7.4V input provided by two 18650 batteries connected in series. Four DC geared motors are controlled using all four channels of the motor driver.

### TT Geared Motors

TT geared motors are small DC motors integrated with a plastic gearbox. The motors are widely used in robotics and hobby electronics. The motors operate typically at 6V to 9V. These motors consist of a brushed DC motor coupled with a plastic gearbox that enhances torque output while reducing speed.

In this project four motors are used to facilitate the movement of the car. Although a stepper motor is not used in this project, the direction of the car can be changed by moving the motors on one side forward and the motors on the other side backward, or vice versa.

## Additional Hardware Components

### Power supply

The main supply for both the controller glove and the vehicle consists of multiple 18650 rechargeable batteries. The name of the battery itself is an indication of its size. It has an 18mm diameter, 65 mm length, and cylindrical shape. The nominal voltage of one cell is 3.7V, while the fully charged voltage is 4.2V.

For the glove controller, one single battery cell is used to power the ESP32 controller through the 3.3V pin. The vehicle uses two cells in series for motor driver power and four cells in parallel inserted into a regulation module, which outputs a constant 5V for other vehicle electronics.

### SSD1306 OLED Display

The SSD1306 OLED Display uses the SSD1306 monochrome display controller to display vairous messages or images. The SSD1306 is a CMOS single-chip OLED driver and controller that is specifically intented for operation on organic or polymer light-emitting diode dot-matrix graphic displays.

The chip includes on-chip display RAM, a contrast control circuitry, and an oscilator which are used to minimize the number of external components needed and the overall power consumption. The device can be communicated with using I2C interface.

For this project, the OLED module is used to display roll and pitch angles on the glove, as well as the connection status of the glove with the vehicle.

### Protection components

In order to ensure the reliability and longevity of the system, some basic protective components have been incorporated into the hardware system.

On the glove power supply, one diode is placed in series with the 18650 battery that drives the glove. The diode provides reverse polarity protection and, should the battery be inserted in reverse, safeguards the ESP32 and other sensitive components. The diode prevents current from flowing in only the correct direction, safeguarding the glove’s electronics. The diode can act as a protection in case the ESP32 is powered by USB for programming, blocking the current flow from the ESP32 to the battery.

Two 2200 uF capacitors are placed in parallel across the motor driver’s input, directly across the power supply lines. The capacitors serve to stabilize the supply voltage by eliminating voltage spikes and electrical noise that are generated when the motors start, stop, or reverse. It avoids unwanted resets or failure of the motor driver and aids more stable and smoother operation of the motors.

Together, these protective components are necessary to provide stable operation and protect the system from common electrical faults.

## Communication Protocols

### Overview

Embedded systems often need to exchange data with other devices, sensors, or modules. This data transfer requires standardized communication protocols, which define the rules for data formatting, timing, and transmission to ensure reliable and efficient communication.

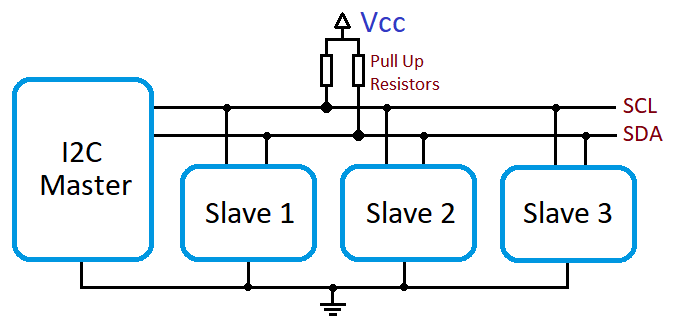
In this project the communication protocols I2C, UART, and ESP-NOW were used, which will be presented in the following chapters.

### I2C Communication Protocol

The Inter-Integrated Circuit Protocol is used to establish the communication between numerous nodes in the system over a short distance. The protocol is designed especially for circuits that are present on the same PCB.

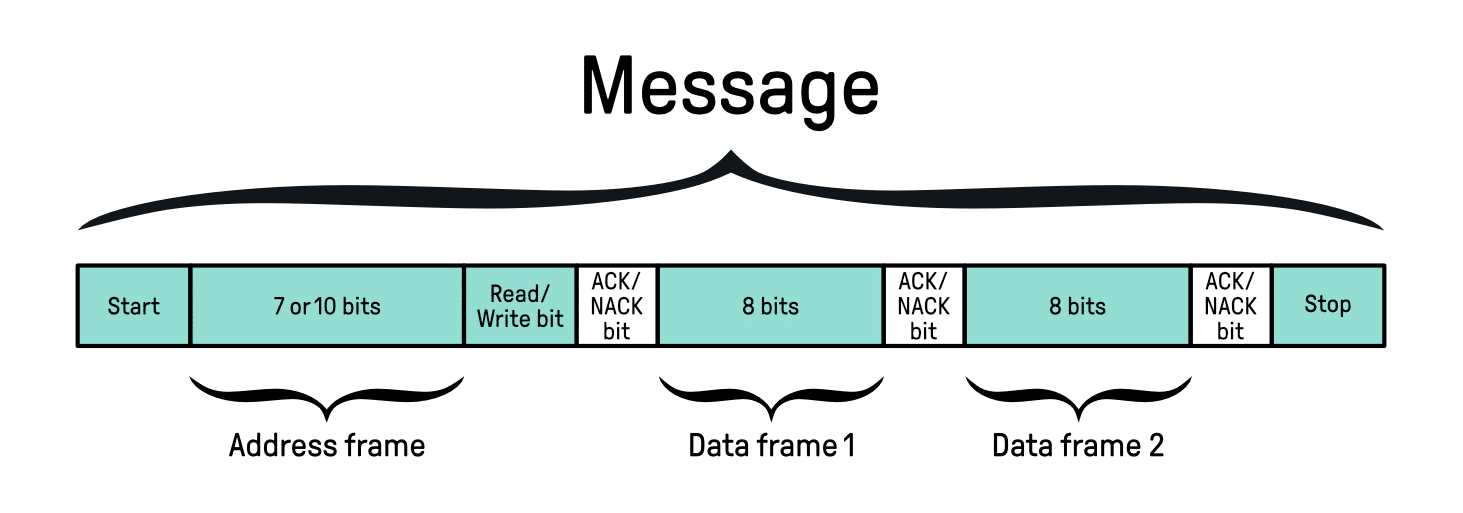
I2C is a master-slave synchronous protocol that allows the master to initiate the communication with the slave. The master provides the clock signal that becomes the data transfer rate. The bus is a bi-directional serial bus, meaning that the master can communicate with the slave and vice versa, and the data is transferred bit by bit every clock pulse.

I2C works with only two wires: SDA (serial data) and SCL (serial clock). Both wires need to be connected to a pull-up resistor to communicate properly. The SLA and SCL buses are open drain, meaning that the transmitting device is pulling the bus down to create logic 0s and letting the bus go to create logic 1s. In the IDLE state, the bus will stay high by the means of the pull-up resitors.[[PROD24](#PROD24)]



The I2C message is composed as follows:

* Address frame – A sequence of bits unique to each connected device on the bus. The master uses this field to target the slave.
* Read/Write bit – Indicates whether the master sends or requests information from the slave device.
* Acknowledge bit – The peripheral device sends the ACK back to the master if the data is successfully acknowledged.
* Data Frames – Once the communication with the slave device is established, the data transfer is initiated.[[SOLD24](#SOLD24)]



I2C communication protocol is mainly used on the controller glove in this project. Its main use is to establish the communication with the MPU9250 accelerometer sensor and SSD1306 OLED. External pull-up resistors are not in this configuration; the MPU9250 and SSD1306 OLED come with built-in pull-up resitors on the PCB.

### UART Communication Protocol

UART, or Universal Asynchronous Receiver/Transmitter is a protocol that manages the transmission of serial data between two devices. It operates over two wires alone – one transmit wire and one receive wire – and a shared ground connection. UART communication can be configured for simplex (one way), half-duplex (each device sends data, but not at the same time), or full-duplex (each device can send and receive data at the same time). The data is sent in formatted units referred to as frames.

One of the most significant benefits of UART is that it is asynchronous (the transmitter and receiver are not using the same clock signal). This makes the protocol easier in general but does place some requirements on things. Because there is no common clock, both units have to be running at the same rate, the baud rate, so that they may synchronize their bit timing.

Digital systems set a high voltage to represent a logical one, whereas low voltage means logical zero. Since the UART protocol does not specify any particular voltage level or range, the high state is called mark, while the low state is called space. When the line is idle, no data transmission takes place, and the bus is in a high-level state. The idle high state makes it easier to check whether the line or transmitter is damaged.

The main parts of a UART packet are as follows:

* Start and stop bits – in asynchronous UART communication, the start of data transmission is signaled by the transmitter. Hence, a start bit goes low, signaling a transition from an idle high state, and is followed by user data bits. The stop bit signals the end of the data frame once data bits are complete. The stop bit either returns the line to a high or idle state or holds the line high for one more bit period.
* Data bits – are the actual user information after the start bit. The data bits are transmitted least significant bit first.
* Parity bit – this bit is optional, and it is used for error detection. The parity bit is placed between the data bits and the stop bit. The value of the parity bit depends on the selected parity scheme. Even parity means that the parity bit is chosen so that the total number of ones in the frame is even, whereas odd parity means that the value of the bit is chosen so that the total number of ones is odd.

The UART protocol is not perfect, and it comes with a big list of limitations and disadvantages such as:

* No built-in clock synchronization – since UART is asynchronous, the transmitter and receiver rely on pre-agreed baud rates rather than a shared clock signal. This makes UART very sensitive to small timing differences. Small differences between devices in baud rate accuracy are sufficient to result in framing errors or mangled data, especially when used for long transmissions.
* Limited error detection – UART has limited error detection, generally through an optional parity bit. Parity also detects single-bit errors but is less useful for multiple-bit errors or more complex faults. It does not handle more active error detection or correction methods like CRC or checksums, so additional software layers might be required for demanding uses.
* No addressing or multipoint support – UART is fundamentally a point-to-point protocol with only two devices to exchange data. It does not natively handle or address multi-device networks like protocols CAN or I2C do.
* Limited data rate and cable length – The maximum dependable baud rate and wire length are limited by electrical noise, wire quality, and signal loss. Higher baud rates limit wire length due to timing accuracy requirements. Normal UART communication is reliable at only a couple of meters without dedicated line drivers or shielding.
* Prone to noise and interference – signals are simple voltage levels and are susceptible to noise, especially in electrically noisy systems or over long cables. Without differential signaling, UART can experience bit errors and data corruption.

This project uses UART configured in simplex mode for communication between the ESP32 microcontroller and Arduino UNO. Only one wire is used, not feedback or response is needed from the Arduino UNO.[[AXE14](#AXE14)]

### ESP-NOW Communication Protocol

ESP-NOW is a wireless communication protocol with low power and without connections, which was designed by Espressif. It enables multiple ESP32 or ESP8266 devices to send small packets of data directly to one another, without depending on Wi-Fi connection or a router.

The protocol is asynchronous, light, and best for short bursts of data such as sensor outputs. It operates at 2.4 GHz; therefore no wires are needed. The communication can be simplex or half-duplex.

The data packet of ESP-NOW is composed as follows:

* MAC address of destination peer.
* Payload, up to 250 bytes on ESP32.
* Optional encryption, using a shared key.
* Status flags such as success or failure callback.

The timing of data transfer is internally handled by the wireless stack. It does not include hardware-level synchronization.

The ESP-NOW protocol is proprietary to Espressif Systems so it can be used only in ESP environments. The MAC addresses of the devices need to be known before initiating the communication between devices.

In this project the protocol is used for MPU9250 data transmission between the controlled glove and the ESP32 present on the car.[[ESP25](#ESP25)]

## Development Tools

### Visual Studio Code

Visual Studio Code (VS Code) is a free, open-source code editor developed by Microsoft Corporation. Developers widely adopt it for its lightweight footprint, high performance, and support for a wide range of programming languages and frameworks in software development. VS Code is built for code editing, having syntax highlighting, Intellisense code completion, debugging support, Git integration, and a rich extension ecosystem as some of its features.

Being an IDE, VS Code is completely customizable; hence, users can tailor the entire development environment with themes, keybindings, or any extensions that might limit their workflow or enhance productivity or support for the tools and platforms they use. With integrated terminal support, a powerful search tool, and being truly multilingual, this entire kit is useful for small scripts or massive, large-scale software projects.

Within the project, my primary development environment was Visual Studio Code. I wrote, edited, and organized embedded software for ESP32 and Arduino UNO boards in this environment. With the help of an extension, I could manage, build, and upload code to the microcontrollers directly from the IDE. It provided me with a convenient and efficient workflow, enabling a well-organized project structure and a more productive development life cycle.

### PlatformIO

PlatformIO is an open-source, business-class embedded development ecosystem that fits seamlessly into contemporary code editors such as Visual Studio Code. PlatformIO has support for many microcontroller platforms, such as Arduino, ESP32, and STM32; hence, it is a very flexible tool for embedded system project development. PlatformIO provides features such as intelligent code completion, project dependency management, a cross-platform build system, and firmware uploading with auto-upload.

One of its biggest strengths is the combined development environment workflow that it offers – allowing developers to write code, compile, deploy code onto a microcontroller, and see serial output from one environment. Because of its native support for a multitude of board definitions and frameworks, it minimizes setting up and maintaining embedded software projects to the bare essentials.

PlatformIO organizes each project into a typical directory structure, separating source files, libraries, and configuration settings for maintainability and scalability. The primary configuration file, *platformio.ini*, defines the target platform, board type, framework, upload settings, and monitor settings. This grants explicit control of the build and upload process as well as quick switching between multiple hardware targets or environments for the same project. Personally, I used this configuration to set custom upload ports and tailor serial monitor baud rates so that they are consistent with the communication settings used by ESP32 and Arduino boards. PlatformIO even supports multiple environments within one project, which proved useful when testing different configurations

I used PlatformIO in Visual Studio Code to manage code development on Arduino UNO and ESP32 boards in my project. It allowed me to compile and flash firmware onto target devices efficiently, and observe real-time data using the integrated serial monitor. This streamlined workflow helped me maintain consistencty on different hardware platforms with less manual setup and configuration work.

### C++ Language

#### Overview of C++ in Embedded Systems

C++ is a statically typed, compiled language with high performance, flexibility, and procedural and object-oriented programming paradigms. It was designed and implemented as an extension of C by Bjarne Stroustrup with the aim of providing enhanced support for program structure and abstraction without losing the low-level efficiency and hardware access provided by C. Over the years, C++ has evolved to be a general-purpose, full-featured language used in a wide variety of domains, ranging from game engines and desktop computing to real-time embedded systems.

One of the strongest features of C++ is that it supports low-level system programming, with low-level memory structure control, resource management, and hardware register access. This suits embedded systems well, where resources are limited and efficiency is the primary consideration. C++ supports high-level abstraction such as classes, inheritance, templates, and exceptions (though the latter are occasionally avoided in embedded systems), with room for clean structure and modularity.

In embedded systems, C++ finds application in a bounded manner, tailored to the limitations of the target platform. Dynamic memory allocation, runtime type information, and exception handling are generally disabled or avoided out of fears for code size, run time, and fault tolerance. Programmers rather practice compile-time polymorphism via templates and deterministic resource management with methods such as RAII. These factors help eliminate excess runtime overhead while still being able to take advantage of the language’s abstraction mechanism.

The zero-cost abstraction principle of C++ is perhaps the most helpful thing it provides in an embedded system context. This philosophy ensures that abstractions (such as templates, inline functions, or classes) incur no cost in terms of performance if they are not being used – enabling high-level code to compile down to very efficient machine code. For example, hardware drivers may be instantiated using templates, with the compiler able to produce optimized code for a particular hardware configuration independent of virtual functions or other runtime mechanisms.

C++ also allows portable and reusable layers of software to be developed. Abstraction interfaces may be used to define hardware-independent APIs, with specific implementations adequate for different microcontrollers or boards. The use of the hardware abstraction method improves code reusability, maintainability, and testability – valuable factors in consumer and industrial embedded applications.

While C remains the underpinning of most embedded platforms, especially in bare-metal programming and legacy, C++ is increasingly being preferred in modern embedded software as systems are becoming complex. Most embedded real-time operating systems, such as FreeRTOS or Zephyr, already have support or are C++ compatible. Besides, with the growing number of 32-bit microcontrollers and greater memory capacities, programmers are able to use C++’s complex features more assertively without taking too much power from the system.

Current C++ has solidified its role even in embedded development further. Language constructs like strongly typed enumerations, lambdas, uniform initialization, constexpr computations, and structured bindings make embedded code more expressive and safer but retain full performance control.

Briefly, C++ is a powerful embedded software development toolkit with low-level control blended with high-level abstraction. In the hands of well-disciplined developers, it enables one to create efficient, maintainable, and scalable firmware on microcontrollers and embedded platforms for a wide range of applications.

The C++ programming language is utilized in this project as the primary programming language for every embedded software component. C++ provides the right structure and flexibility for straightforward communication with hardware, real-time data processing, and modular coding. Functions are implemented in the project to carry out tasks like sensor data acquisition, wireless data communication, and motor driving.

#### C++ History

C++ was created in the early 1980s by Bjarne Stroustrup at Bell Laboratories while he was trying to introduce features to the C language that would permit more complex software to be built. The language, which was initially called „C with Classes”, added object-oriented features such as classes, inheritance, and encapsulation on top of the procedural C foundation. This early work was the starting point for what would become C++.

The first commercial release of C++ was in 1985, with the publication of Stroustrup’s comprehensive book „The C++ Programming Language”. Over the next twenty years, the language went through a series of standardizations:

* C++98 was the first ISO standard version, which introduced templates, exceptions, namespaces, and the Standard Template Library.
* C++03 included mostly bug fixes and some minor improvements over C++98.
* C++11 was a major upgrade that introduced modern programming conveniences like lambda functions, auto type deduction, nullptr, smart pointers, and concurrency support.
* C++14, C++17, and C++20 continued to enhance the language with compile-time programming, structured bindings, modules, concepts, and more.

C++ now is a multi-paradigm language – it accommodates procedural, object-oriented, generic, and even functional programming styles – and is widely used for both systems programming and application-level programming.

#### C++ Architecture

The architecture of C++ is dictated by its objectives: flexibility, efficiency, and performance at the expense of abstraction. Its organization can be described on the backdrop of its character as a language, its compilation model, and its runtime behavior.

C++ implements the following layers and paradigms:

* Procedural layer – functions, loops, conditionals, pointers, and explicit memory management.
* Object-oriented layer – encapsulation and code reuse are facilitated using classes, inheritance, and polymorphism.
* Generic layer – templates give compile-time polymorphism, necessary for type-safe generic programming.
* Functional layer – lambdas, closures, and functions enable functional programming abstractions.

C++ programming is compiled ahead-of-time using a compiler like GCC, Clang, or MSVC. The process of compilation includes:

* Preprocessing – adds headers and expands macros.
* Complilation – compiles code into machine-level object files.
* Linking – external symbols and libraries get resolved to produce a final executable or firmware image. [[STR13](#STR13)][[BAR06](#BAR06)]

## Software Libraries

Software libraries are used not only in embedded systems, but also in various types of applications. The main role of a software library is to reduce development time, making the code more reliable. Software libraries standardize the behavior of diverse hardware platforms. A library is a collection of pre-written code – typically packaged in functions, classes, or modules – that can be reused in multiple programs.

In embedded software development, libraries provide basic services such as hardware abstraction, peripheral control, communication protocols, math functions, and even real-time system services.

In the present project various software libraries were used to ease the process of development. All libraries all presented in the following chapters.

### AFMotor

The AFMotor library is a software library from Adafruit that can be employed with the Adafruit Motor Shield, which is a very popular add-on shield for Arduino boards. The library provides an easy to-use and simple interface for managing DC motors, stepper motors, and servos on the shield. It conceals the motor control low-level specifics, and users can easily specify motor speed, direction, and running mode through simple function calls. The AFMotor library is widely used in robotics and automation applications due to the simplicity of usage as well as the capacity to interface with a wide variety of motors.

In this project, the library is being employed on the Arduino UNO as the motor driver of the remote-controlled car. The library is being used to operate the DC motors employed in moving the car. Instructions sent from the ESP32 controller are received by the Arduino UNO, and it employs the AFMotor library to set the correct speed and direction for each motor. This enables the vehicle to move forward, reverse, turn left, turn right, or brake, respectively, based on the user’s hand movements detected by the glove. Use of the AFMotor library simplifies implementing the motor control logic and ensures effective and precise actuation of the vehicle’s motors.

### Adafruit\_GFX and Adafruit\_SSD1306

Adafruit\_GFX is a general graphics core library by Adafruit Industries. It is a set of general- purpose graphics primitives, including functions for drawing pixels, lines, rectangles, circles, and bitmaps. It also supports text rendering, allowing the writing of strings and characters in various fonts, sizes, and alignments. One of its major strengths is hardware abstraction; the library is designed to work in conjunction with display-specific driver libraries, so the same graphics calls will work regardless of display type. This modularity allows the developer to focus on user interface development rather than the specifics of the display.

The Adafruit\_SSD1306 software library is a display driver library that is designed to work specifically with OLED displays that use the SSD1306 controller chip. The library handles the low-level communication between the microcontroller and the OLED display chip, working with both I2C and SPI interfaces. It provides functionality to initialize the display, set the brightness, clear the display, and shove pixel data into the display buffer.

Those libraries are companion libraries; Adafruit\_GFX handles the actual drawing and text printing, whereas Adafruit\_SSD1306 does the actual hardware communication and ensures the graphics output of the Adafruit\_GFX is rendered correctly.

In this project, the application of Adafruit\_GFX with Adafruit\_SSD1306 libraries enables the system to show real-time visual output to the user via the OLED display. The Adafruit\_SSD1306 library handles the initialization and communication of the OLED hardware, while Adafruit\_GFX is used for text and graphics rendering, such as system connection status and sensor data. This double-layer architecture allows clean, readable, and visually apparent information to be presented, facilitating the usability and interactivity of the overall system.

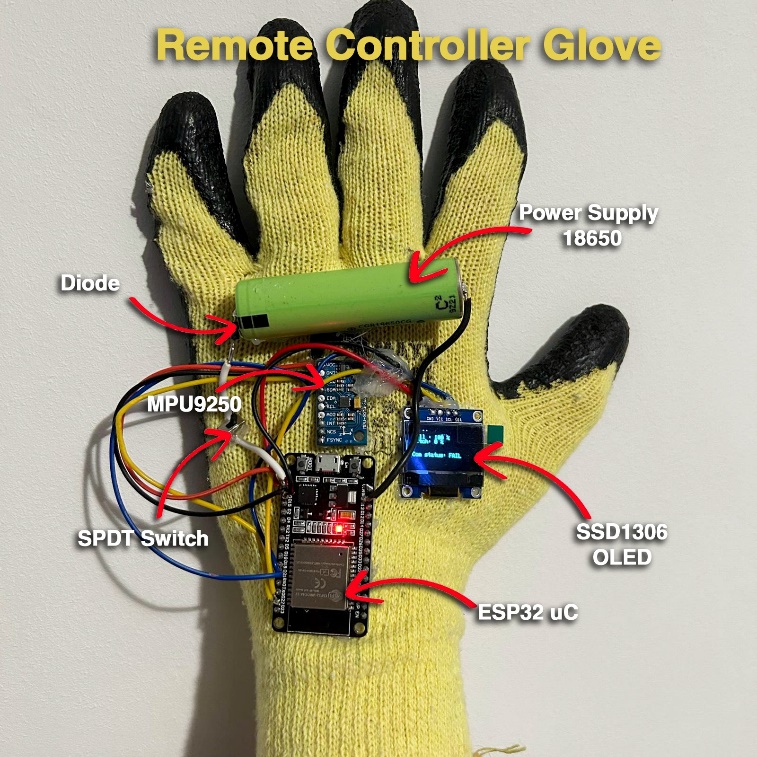
# PROJECT IMPLEMENTATION

## Implementation Overview

This project involves the design and implementation of a remote-controlled vehicle system that is operated through a wereable glove-based controller. This implementation is divided into two core components: the remove controller glove, which captures user gestures and translates them into inclination angles, and the remote-controlled vehicle, which receives those angles, transforms the raw data into driving commands and executes corresponding movements.

## Remote Controller Glove

### Hardware Implementation



#### Hardware Connections

The hardware of the remote controller glove integrates several embedded components mounted on a protective textile glove. These components are interconnected to form a functional system capable of detecting hand gestures and transmitting them wirelessly to a remote-controlled vehicle.

The ESP32 Microcontroller Unit acts as the brain of the remote controller glove, responsible for capturing sensor information from the IMU MPU9250, displaying information on the OLED display, and transferring data to the remote-controlled vehicle through wireless communication.

The SSD1306 OLED and MPU9250 accelerometer sensor communicate with the ESP32 microcontroller unit using an I2C interface. All connections are soldered on the physical pins of the integrated circuits in order to create strong connections. All circuits are mounted on the glove using a silicone hot glue gun. This method provides both insulation and mechanical stability without damaging the textile surface.

#### Power Supply

The power supply of the remote controller glove is based on a single 18650 Li-ion battery. The battery is chosen for its high energy density, compact form factor, and rechargeable nature, making it ideal for wearable electronics.

The battery provides a nominal voltage of 3.7V which is perfectly in range for supplying the ESP32 through the 3.3V pin. The battery has a capacity of approximately 2000 mAh, this is sufficient to power the glove for several hours.

The battery is placed in series with a protection diode, connecting to all other circuits on the glove. The protection diode is placed on the positive terminal of the battery, preventing permanent damage in the event of wiring mistakes or sudden power fluctuations.

The glove is equipped with a SPDT switch to manually disconnect the power. This switch enables the user to turn off the system to save battery when not in use.

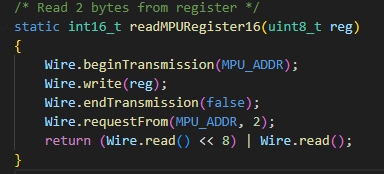
### Software Implementation

The software running on the remote glove is responsible for capturing the user’s hand movement, processing this data, providing real-time feedback, and wirelessly transmitting inclination angles to the vehicle. Each module handles a specific function to ensure clarity, maintainability, and responsiveness.

#### Sensor Readings

The Remote\_Glove software block handles sensor reading through the *gyro.cpp* software module. This module is responsible for initializing the IMU MPU9250 accelerometer, gyroscope, and magnetometer sensor and reading its information cyclically from the registers. Communication with the module is realized by using the I2C interface provided by the *Wire.h* library.

MPU9250 accelerometer data can be accessed by performing 16-bit read operations on registers 0x3B (X-axis data), 0x3D (Y-axis data), and 0x3F (Z-axis data). For this, a special function dedicated to reading 16-bit registers was created.



The function *readMPURegister16* is called especially for reading accelerometer data raw register values.

After raw register data is read from the registers, a series of mathematical formulas are applied in order to transform raw data into actual roll and pitch inclination percentages.

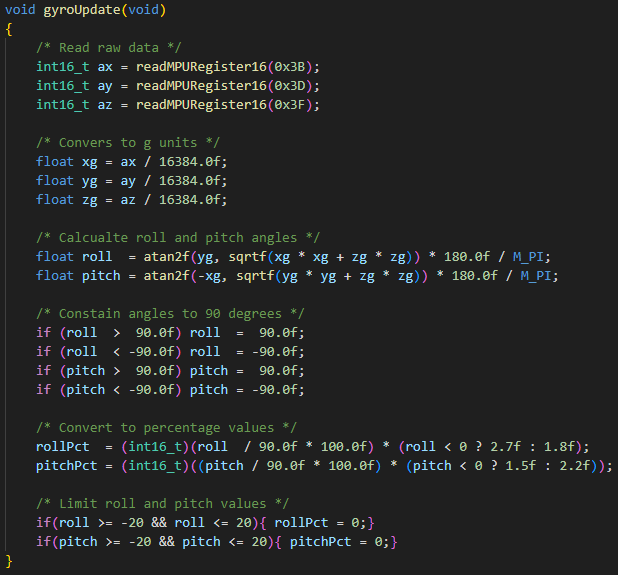
The first step is transforming raw data into *g-units*, which refer to the acceleration relative to the acceleration due to gravity on Earth. For the sensor raw data, 16384 corresponds to 1g; thus, by applying the following formula, we can transform the register values into real *g units*.

The next step is to determine the roll and pitch of the glove. Roll is the rotation of the sensor around its X-axis (left or right), and pitch is the rotation around the Y-axis (forward or backward). These angles describe the orientation of the glove in 3D space and are essential for translating hand gestures into movement commands for the vehicle. First, we need to determine the combined acceleration in the plane perpendicular to the remaining axis, which is determined by the following formula.

In order to determine the tilt ratio we need to apply the following formulas:

And to determine the angle in radians and transform it into degrees we can apply the formula:

Those angles will be transformed to percentages representing the amount of tilt for all directions. Those formulas are applied in code in the function *gyroUpdate*.



Percentages are the output of the *gyro.cpp* software component can be accessed through two getter functions, *gyroGetRollPct* and *gyroGetPitchPct.*

#### Display Information

The OLED display is handled in the oled.cpp software module, where only two functions are implemented: *oledSetup* and *oledShowInclination*.

The OLED display is initialized in function *oledSetup* where methods borrowed from the Adafruit\_SSD1306 and Adafruit\_GFX libraries are used. The method *begin* is used to specify the power source for the display and OLED physical I2C address. The color of the display is set to white, and the text size is set as one. In case the initialization of the OLED display fails, the whole ESP32 system is restarted in an attempt to fix the connection.



The function *oledShowInclination* is responsible for displaying the inclination angles captured by the MPU9250 sensor and the communication status of the ESP32 from the controller glove to the ESP32 of the remote-controlled vehicle. Simple methods from the Adafruit libraries are used to display the text, like *print* or *println*. Connection status is received from the *wifi\_control.cpp* software module, and the display is set to blink in order to warn the user of the connection status.



#### Communication Module

The wireless communication is realized using ESP-NEW proprietary communication protocol.

## Remote-Controlled Vehicle

### Hardware Implementation

#### Hardware Connections

#### Power Supply

### Software Implementation

#### Vehicle Control Unit

##### Sensor Readings

##### Drive Command Decision

##### Communication Module

#### Motor Driver

##### Motor Control Logic

##### Speed and Direction Handling

## System Testing and Evaluation

### Testing Procedure

### Observations

### Limitations and Challanges

### Testing Conclusions

# Concluzii

Autorul prezintă concluziile sale…

# Bibliografie

**[****AXE14]** – Axelson, J. (2014). Serial Port Complete: COM Ports, USB Virtual COM Ports, and Ports for Embedded Systems. Lakeview Research.

**[BAR06]** – Barr, M., & Massa, A. (2006). Programming Embedded Systems (2nd ed.). O’Reilly Media.

**[LIN20]** – Lin, C.-S., Chen, P.-C., Pan, Y.-C., Chang, C.-M., & Huang, K.-L. (2020). The manipulation of real-time Kinect-based robotic arm using double-hand gestures. Journal of Sensors, 2020, Article ID 9270829.

**[MAZ13]** – Mazidi, M.A., Naimi, S., & Naimi, S. (2013). The 8051 Microcontroller and Embedded Systems: Using Assembly and C for ATMEL. Pearson.

[STR13] – Stroustrup, B. (2013). The C++ Programming Language (4th Edition). Addison-Wesley.

# Referințe web

**[ARDU23]** – Arduino, Arduino UNO Rev3 Datasheet, 2023. Available at: <https://content.arduino.cc/assets/UNO-TH_Rev3e_SCH.pdf>

**[ALLA21]** – All About Circuits, Introduction to the Capacitive Accelerometer: Measure Acceleration with Capacitive Sensing. Available at: <https://www.allaboutcircuits.com/technical-articles/introduction-to-capacitive-accelerometer-measure-acceleration-capacitive-sensing/> [Accessed: 8 June 2025].

**[ALLG22]** – All About Circuits, Introduction to MEMS Gyroscopes: Vibratory Gyroscope. Available at: <https://www.allaboutcircuits.com/technical-articles/introduction-to-mems-gyroscopes-vibratory-gyroscope/> [Accessed: 8 June 2025].

**[ALLM21]** – All About Circuits, Types of Magnetometers. Available at: <https://www.allaboutcircuits.com/technical-articles/types-of-magnetometers/> [Accessed: 8 June 2025].

**[ENPR17]** – The Engineering Projects, Introduction to ATmega328, 2017. Available at: <https://www.theengineeringprojects.com/2017/08/introduction-to-atmega328.html> [Accessed: 8 June 2025].

**[ESPD23]** – Espressif Systems, ESP32 Series Datasheet, Version 4.3, 2023. Available at: <https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf>

**[****ESP25]** – Espressif Systems. (n.d.). ESP-NOW API Reference (ESP32).  
<https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_now.html>

**[ESPT23]** – Espressif Systems, ESP32 Technical Reference and Documentation Portal, 2023. Available at: <https://www.espressif.com/en/support/documents/technical-documents>

**[IN****VE13]** – InvenSense, MPU-9250 Product Specification, Rev. 1.1, 2013. Available at: <https://www.invensense.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf>

**[KEYE20]** – KEYENCE, What is an Ultrasonic Sensor? Sensor Basics (Detection based on Ultrasonic Waves). Available at: <https://www.keyence.com/ss/products/sensor/sensorbasics/ultrasonic/info/> [Accessed: 8 June 2025].

**[MICR15]** – Microchip Technology Inc., ATmega328/P 8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash, Rev. 7810D–AVR–01/15, 2015. Available at: <https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf>

**[PROD24]** – Prodigy Technovations, I2C Protocol. Available at: <https://www.prodigytechno.com/i2c-protocol> [Accessed: 8 June 2025].

**[ROBO23]** – Robocraze, What is an Ultrasonic Sensor?. Available at: <https://robocraze.com/blogs/post/what-is-ultrasonic-sensor> [Accessed: 8 June 2025].

**[SOLD24]** – Soldered, What is the I2C Communication Protocol?. Available at: <https://soldered.com/learn/what-is-the-i2c-communication-protocol/> [Accessed: 8 June 2025].

**[TEXA19]** – Texas Instruments, L293, L293D Quadruple Half-H Drivers Datasheet, Rev. S, October 2019. Available at: <https://www.ti.com/lit/ds/symlink/l293.pdf>

# Codul sursă

În această anexă se adaugă codul sursă al aplicației…

# Site-ul web al proiectului

Autorul prezintă în această anexă (opțională) site-ul web asociat proiectului său.

# CD / DVD

Autorul atașează în această anexă obligatorie, versiunea electronică a aplicației, a acestei lucrări, precum și prezentarea finală a tezei.



# Index

B

Bibliografie 9

C

CUPRINSUL xi

D

Dimensiuni 3

F

Figuri 4

Formulele matematice 4

I

Ilustrațiile 4

L

Legenda 6

LISTA FIGURILOR xii

LISTA TABELELOR xiii

R

Referințe web 10

S

Structura documentului 2

T

Tabele 5