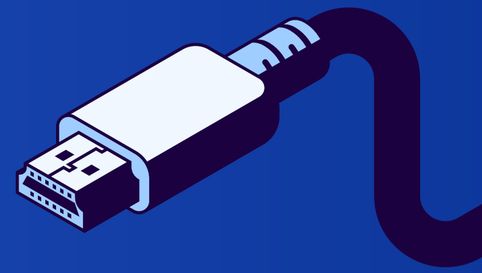
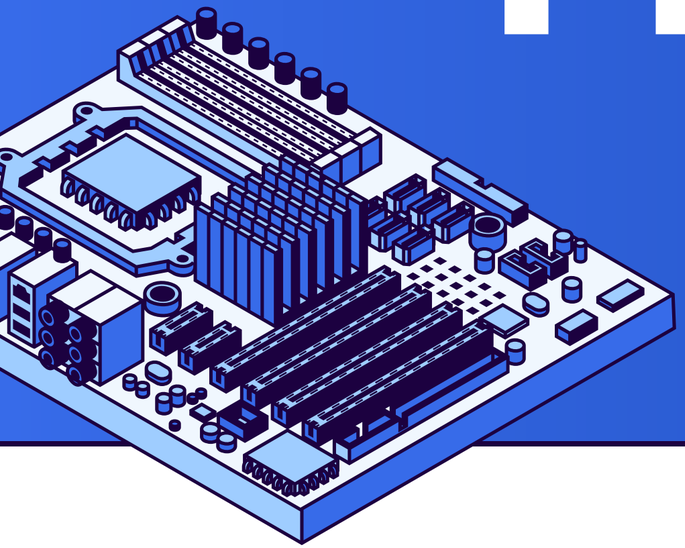
Ziad Fathy

2025

Scout Robot Presentation





EMBEDDED SYSTEM PROJECT

ZIAD FATHY

Ziad Fathy

2025

Agenda

01

Project Idea

02

Solidworks Design & Matlab modeling

03

Hardware Architecture & Conections

04

Software Architecture & connections

05

Robotic operating system

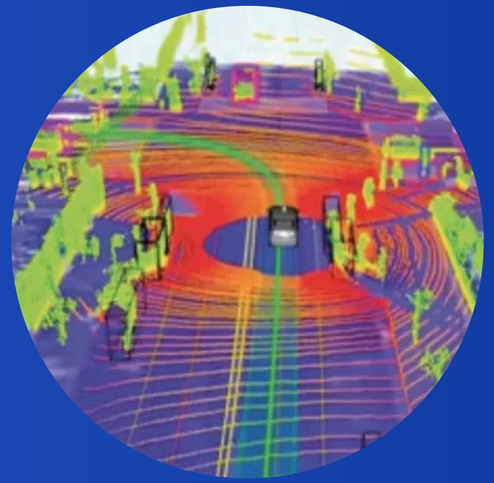
06

TinyML

Project Problem Statements

Inaccessible Confined Spaces

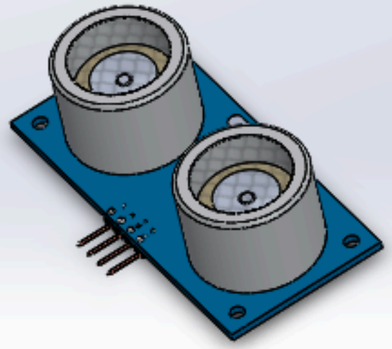
Lack of Real-Time Vision in High-Risk Zones

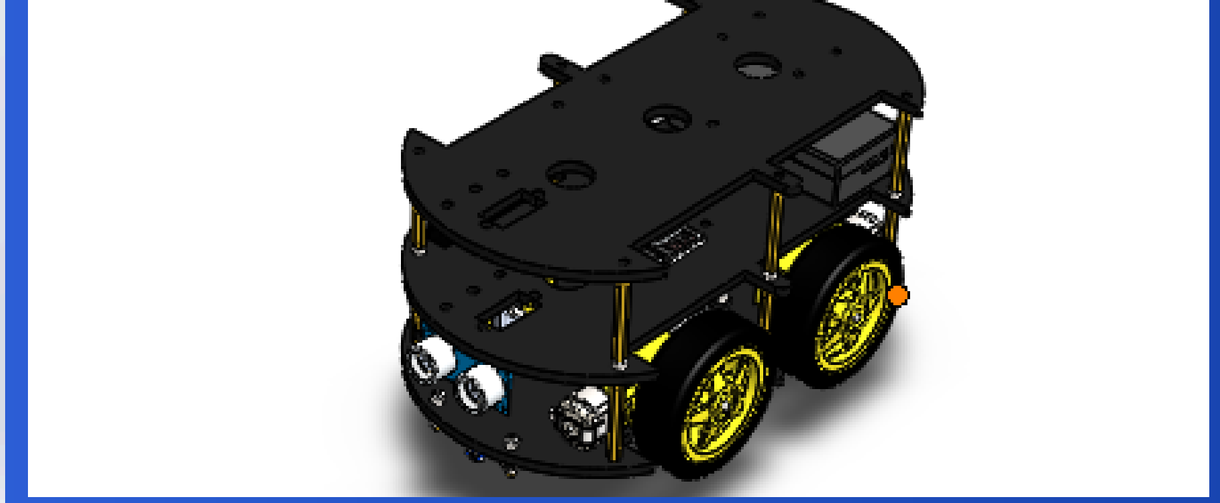


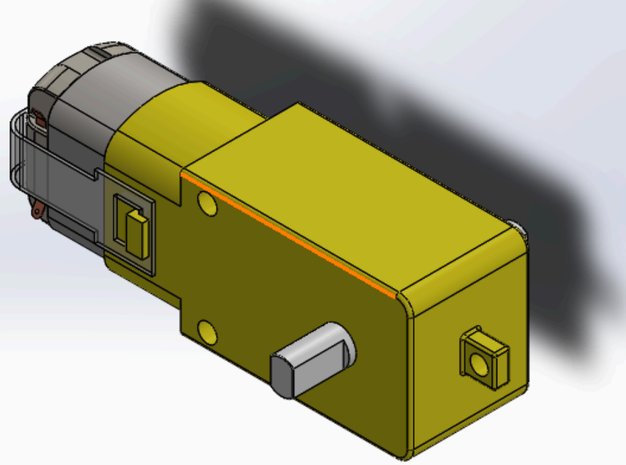
Dynamic Mapping and Localization in Unknown Areas

Idea of project

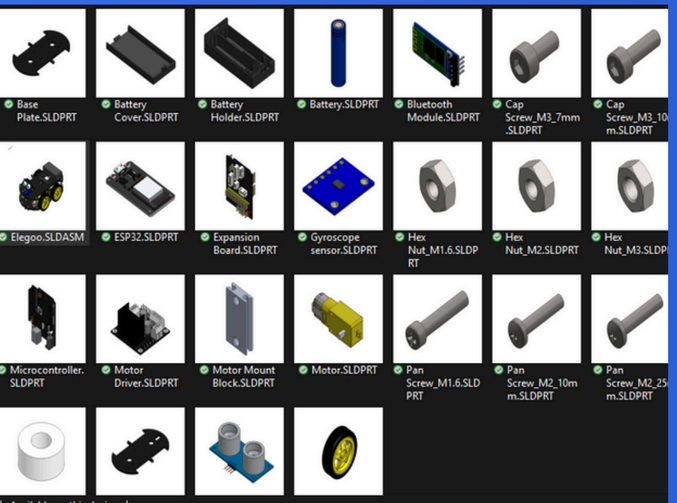
The Autonomous IoT Micro-Explorer is an ESP32-driven miniature vehicle designed for remote navigation and mapping of hazardous or confined environments. It employs MQTT for real-time control via a Mosquitto broker and utilizes RViz2/SLAM to generate dynamic, high-fidelity maps. Integrated ultrasonic and IR sensors ensure precise obstacle avoidance. This solution provides a critical, scalable platform for remote exploration, inspection, and data acquisition in challenging scenarios like disaster response.

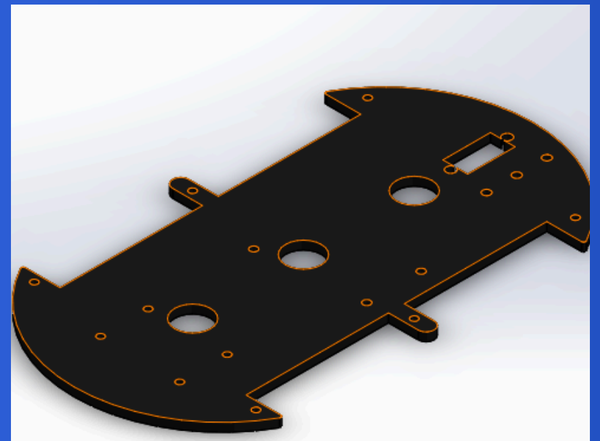


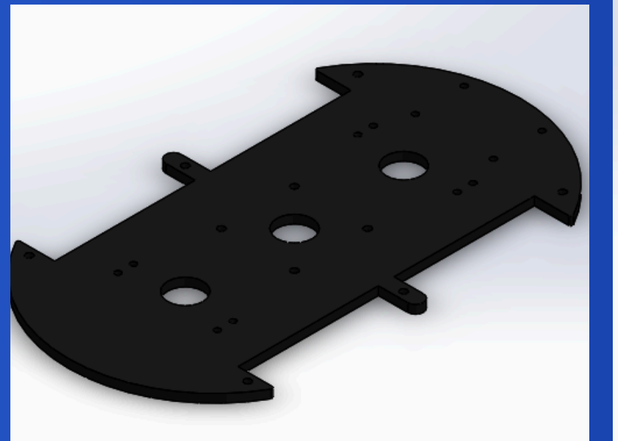


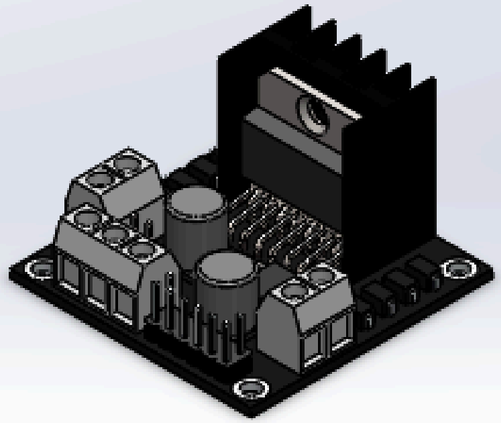


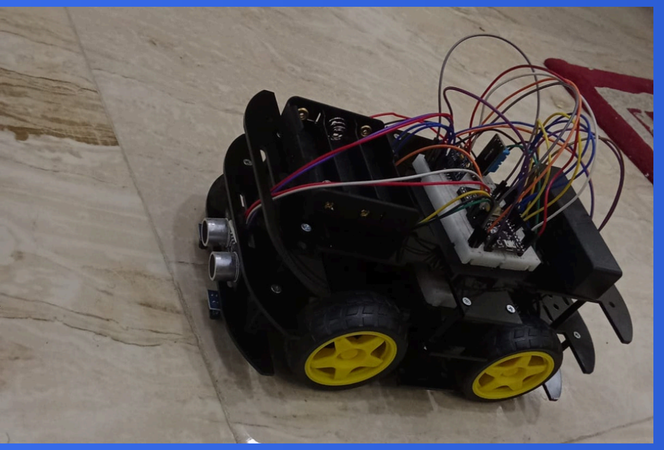
SolidWorks Design

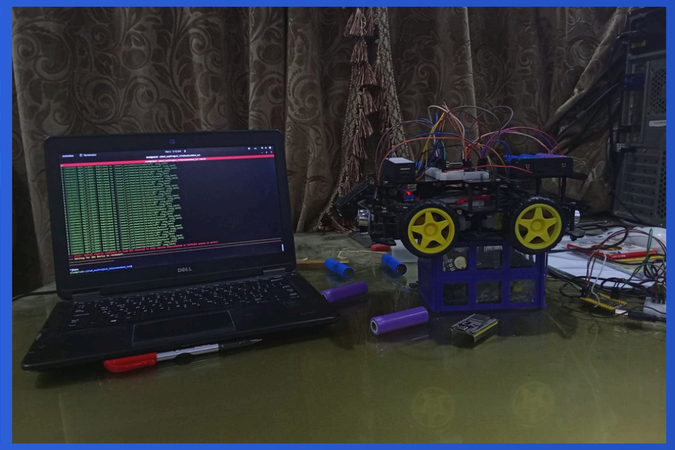


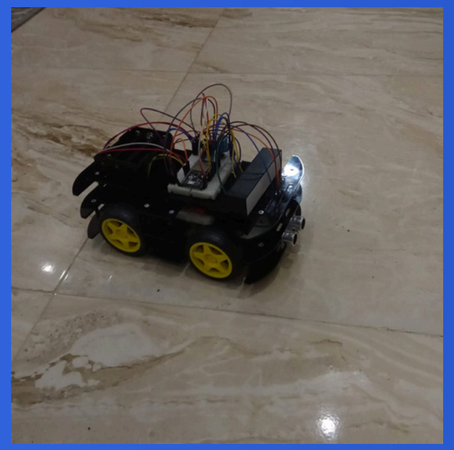


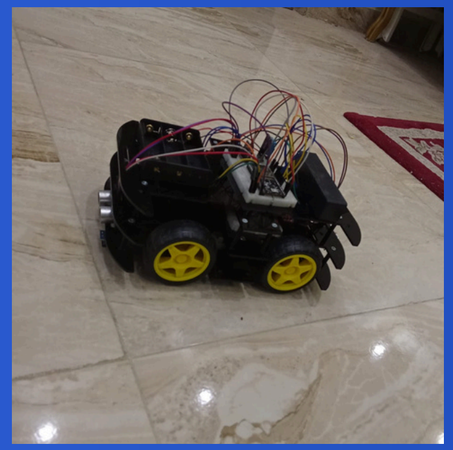






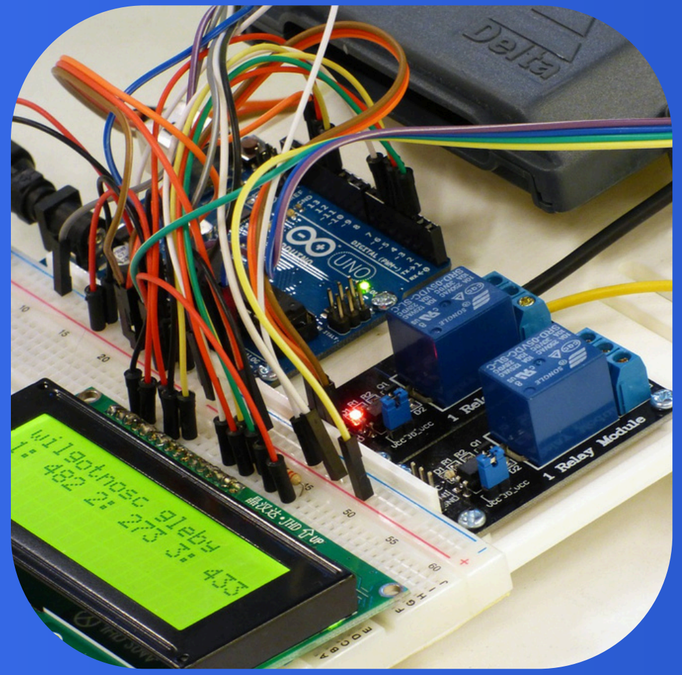






Final Design

Hardware components



01 Micro-Controller ESP32

02 H-Bridge L298

03 DC Motors

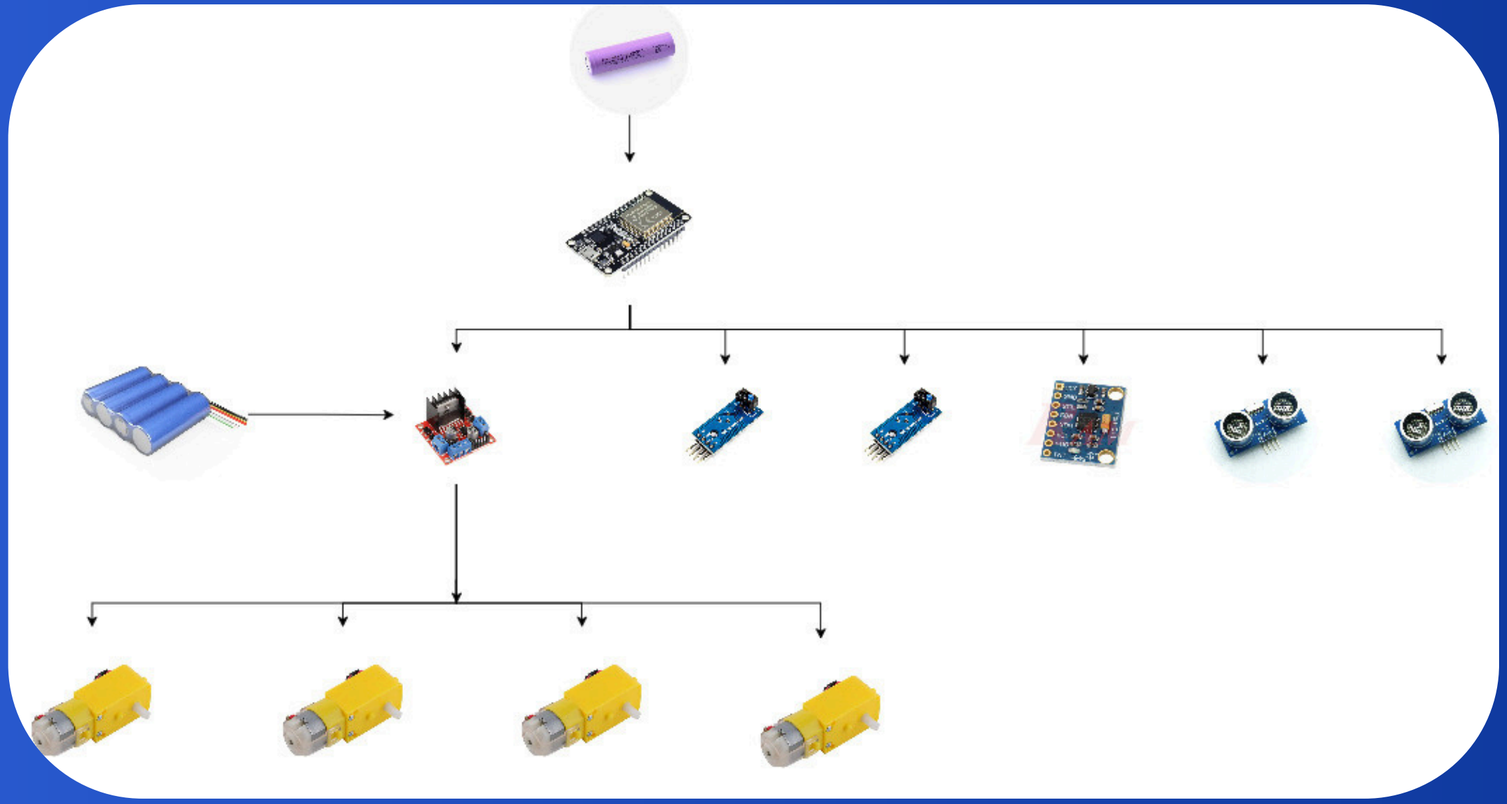
04 Ultrasonic sensors

05 TCRT5000 Sensor

06 lithium battery

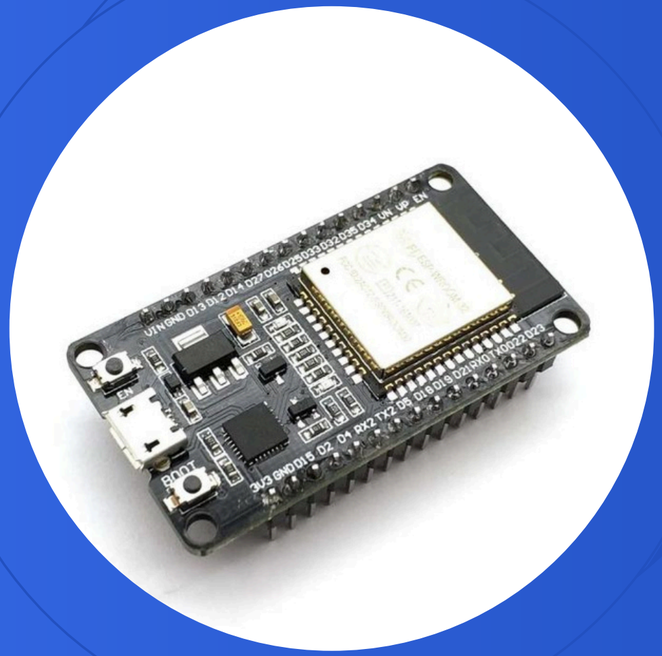
07 MPU6050

Hardware Connections



Ziad Fathy

2025

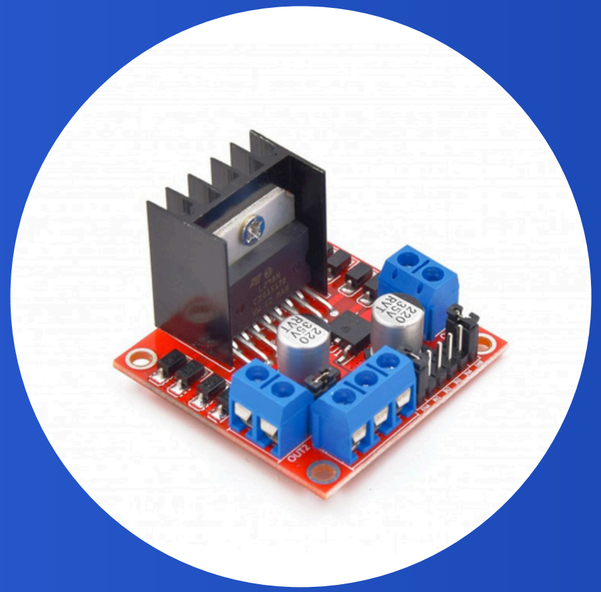


Micro-Contrroller

ESP32 Dev Kit

IT OFFERS WI-FI AND BLUETOOTH CAPABILITIES, ESSENTIAL FOR IOT COMMUNICATION (MQTT/ MOSQUITTO) AND REMOTE CONTROL.
IT ACTS AS THE CENTRAL PROCESSING UNIT, READING DATA FROM ALL SENSORS (ULTRASONIC, IR), PROCESSING IT, AND CONTROLLING THE MOTOR DRIVER (L298) AND DC MOTORS, ALL WHILE RELAYING INFORMATION FOR THE RVIZ2 SLAM SYSTEM.
THIS SINGLE, LOW-POWER, COST-EFFECTIVE CHIP HANDLES ALL THE NECESSARY SENSING, CONTROL, AND WIRELESS NETWORKING FUNCTIONS. (198 CHARACTERS)

Hardware outputs



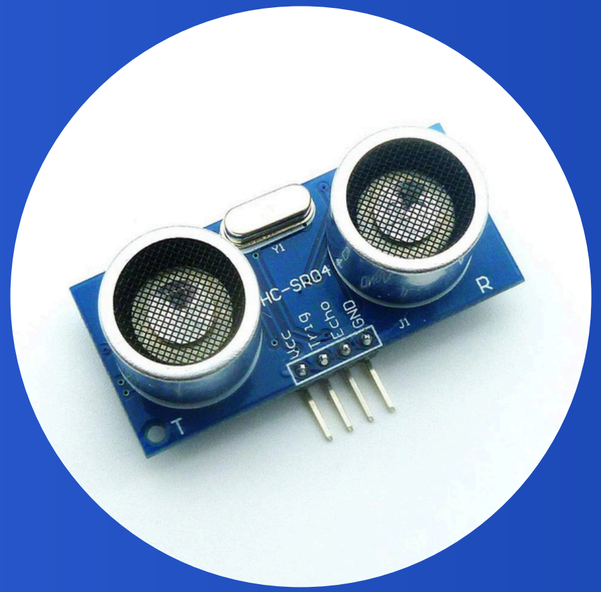
H-bridge (l298)

DRIVES AND CONTROLS THE SPEED AND DIRECTION OF THE DC MOTORS BASED ON SIGNALS FROM THE CONTROLLER.

DC motors

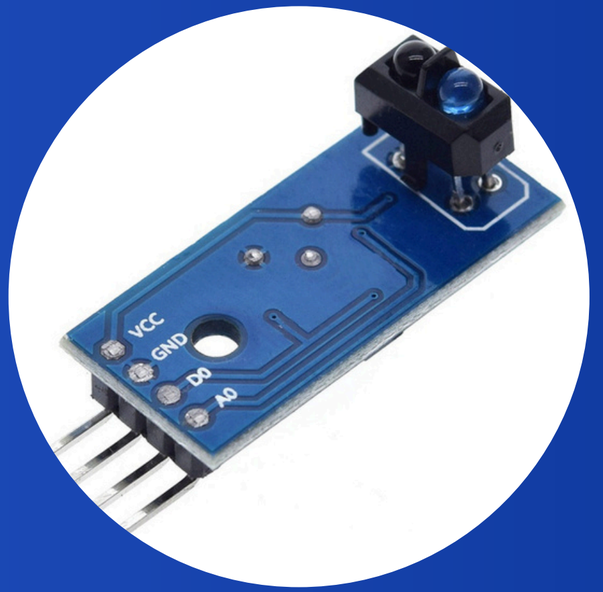
PROVIDES PROPULSION AND DRIVE, ENABLING THE MICRO-EXPLORER'S MOVEMENT AND NAVIGATION.

Hardware inputs



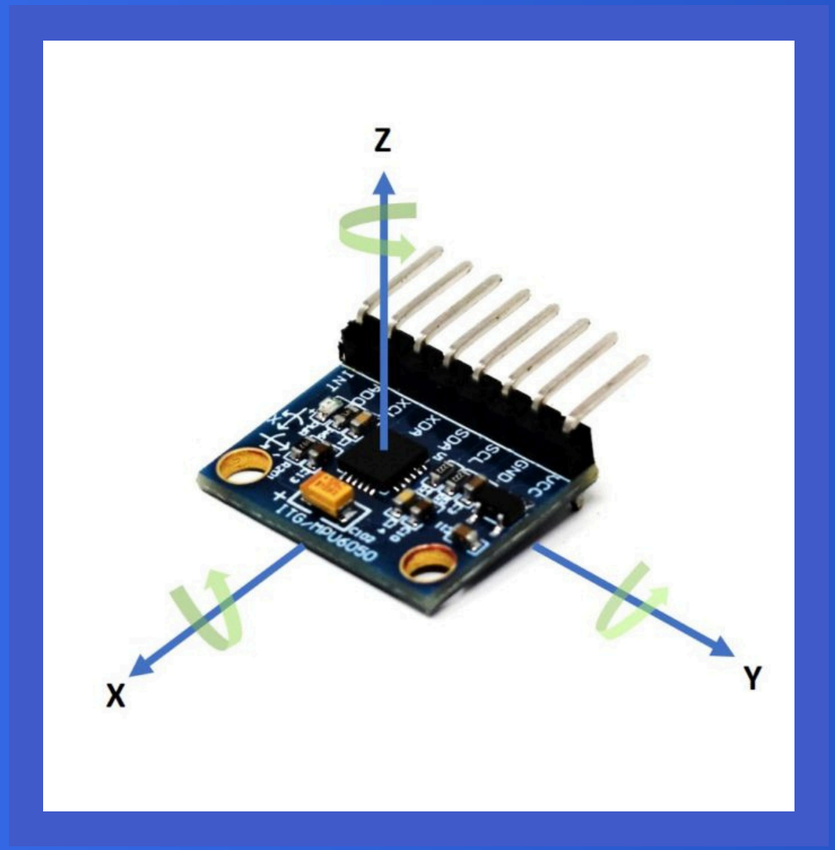
Ultrasonic sensor

MEASURE DISTANCE TO OBSTACLES AHEAD AND BEHIND FOR ANTI-COLLISION AND SAFE NAVIGATION.



TCRT5000 sensor

DETECT NEARBY OBSTACLES OR TRACK LINES FOR CLOSE-RANGE AVOIDANCE AND PRECISE WALL/ LINE FOLLOWING.



MPU6050

I utilize the MPU-6050 (connected via the I2C protocol) to provide essential Inertial Measurement Data (IMU). The integrated Gyroscope and Accelerometer supply real-time angular velocity and acceleration, which are crucial for improving SLAM (Simultaneous Localization and Mapping) accuracy and dead reckoning. This data is essential for correcting for wheel slip and ensuring motion stability in challenging, uneven environments. The integrated temperature sensor is used for device calibration.

Project Software

01 Programming Language

02 Build Tool

03 FrameWork ESP-IDF

04 Layer Architecture

05 Node-Red

06 Mqtt Broker mosquito

Modern C++

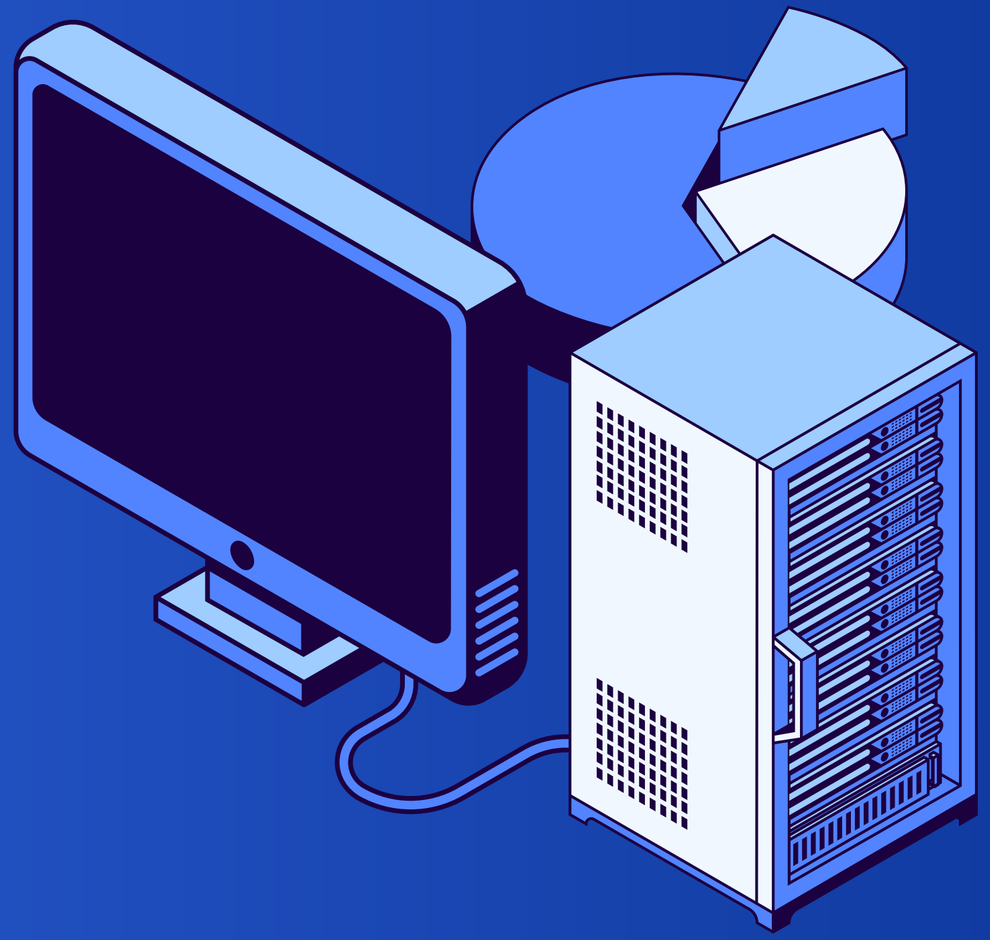
Programming Language

I CHOSE MODERN C++ OVER C OR MATLAB BECAUSE C++ OFFERS POWERFUL OBJECT-ORIENTED PROGRAMMING (OOP) FEATURES, WHICH ARE IDEAL FOR MANAGING COMPLEXITY IN AN EMBEDDED IOT PROJECT. OOP ALLOWS US TO ENCAPSULATE HARDWARE LOGIC INTO CLEAR CLASSES LIKE WIFIMANAGER, MQTTCLIENT, AND DC_MOTOR, MAKING THE CODE MODULAR, REUSABLE, AND EASIER TO MAINTAIN. FURTHERMORE, C++ PROVIDES SUPERIOR TYPE SAFETY AND FEATURES LIKE LAMBDAS (USED FOR EVENT CALLBACKS IN WIFIMANAGER AND MQTTCLIENT) AND STANDARD TEMPLATE LIBRARY (STL) CONTAINERS (STD::VECTOR IN DC_MOTOR, STD::STRING) WHICH SIGNIFICANTLY ENHANCE DEVELOPMENT SPEED AND CODE READABILITY COMPARED TO BARE C. MATLAB, WHILE EXCELLENT FOR SIGNAL PROCESSING AND SIMULATION, IS UNSUITABLE FOR REAL-TIME, RESOURCE-CONSTRAINED EMBEDDED DEVELOPMENT ON THE ESP32. MODERN C++ PROVIDES THE NECESSARY PERFORMANCE, HARDWARE CONTROL, AND ABSTRACTION FOR THIS KIND OF APPLICATION.

Build Tools

CMAKE BUILD TOOL

i utilize CMake as the core build system because it's the standard for ESP-IDF projects. CMake is essential for cross-platform compilation, defining the project structure, and correctly linking our application code (main.cpp, dc_motor.cpp, etc.) with the necessary ESP-IDF components (like esp_wifi, freertos, and nvs_flash). The idf_component_register function specifically tells the toolchain which source files, header directories, and required dependencies must be included. This modularity simplifies management of complex libraries and dependencies, paving the way for the robust features provided by the ESP-IDF we'll discuss next.



Frame Work

ESP-IDF

IS THE OFFICIAL DEVELOPMENT FRAMEWORK FOR THE ESP32. IT PROVIDES THE ESSENTIAL TOOLS, LIBRARIES, AND FREERTOS OPERATING SYSTEM THAT ALLOW US TO WRITE AND RUN LOW-LEVEL C/C++ CODE. IT HANDLES CRITICAL EMBEDDED TASKS LIKE GPIO CONTROL, NETWORKING (WI-FI), PERIPHERAL DRIVERS (E.G., I2C, LEDC), AND NON-VOLATILE STORAGE (NVS). IT FORMS THE FOUNDATION FOR OUR APPLICATION'S HARDWARE INTERACTION AND NETWORKING STACK.

Layer Architecture

01

This layer, often called MCAL (Microcontroller Abstraction Layer), provides direct access and low-level drivers to the ESP32's core hardware peripherals

MCU (Microcontroller Unit)

02

The HAL simplifies direct hardware interaction by offering uniform, high-level APIs for components like motors and sensors (Ultrasonic, IR, etc.).

HALL (Hardware Abstraction Layer)

03

The library layer contains reusable, application-agnostic software components, primarily standard C/C++ utilities and custom drivers like WiFiManager and MQTTClient.

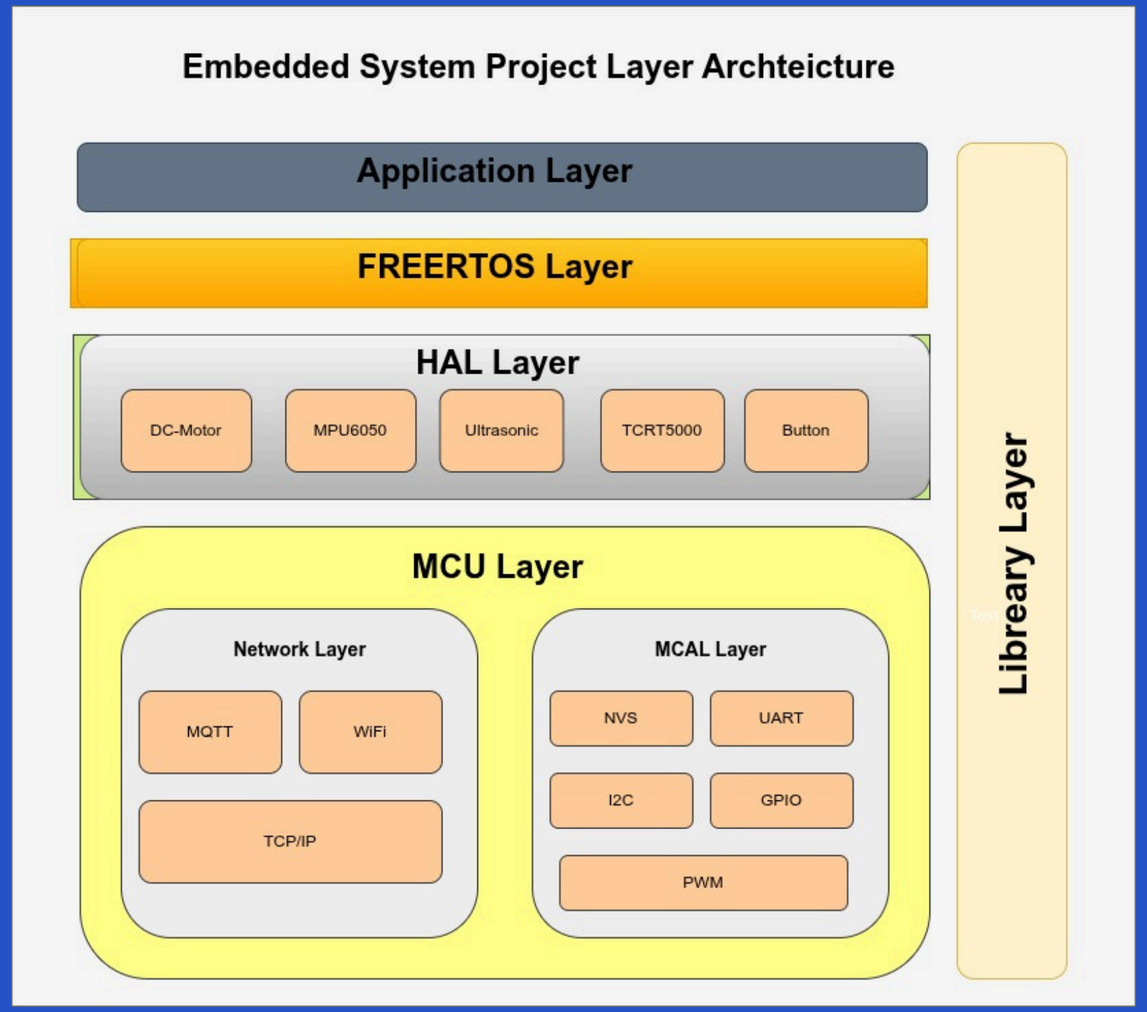
LIB (Library)

04

This is the top-level application logic. It orchestrates the system, reads sensors, processes MQTT commands, and controls the vehicle movement.

MAIN (Application)

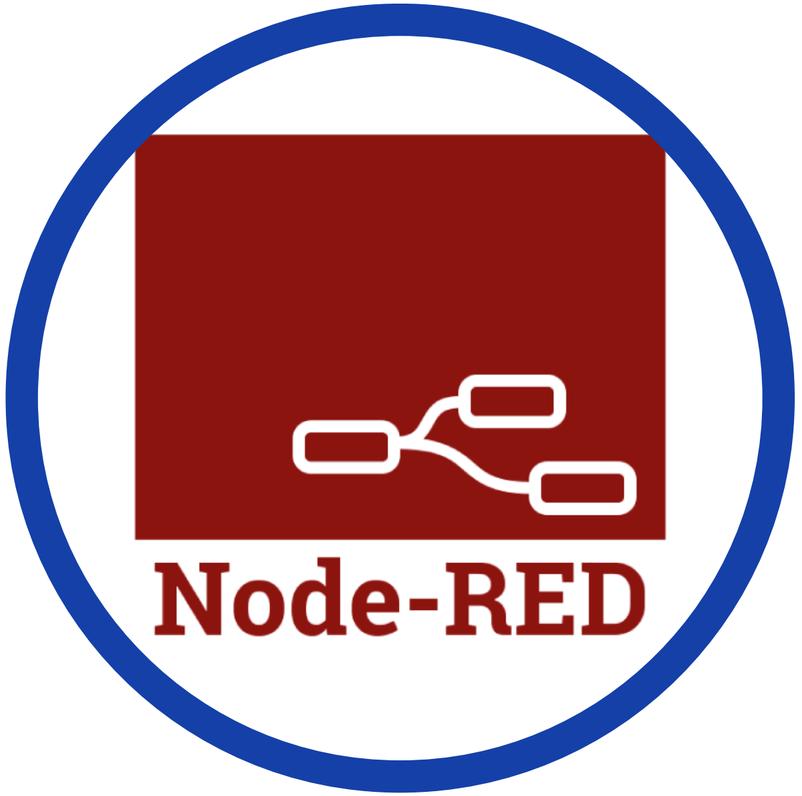
Layer Architecture Diagram



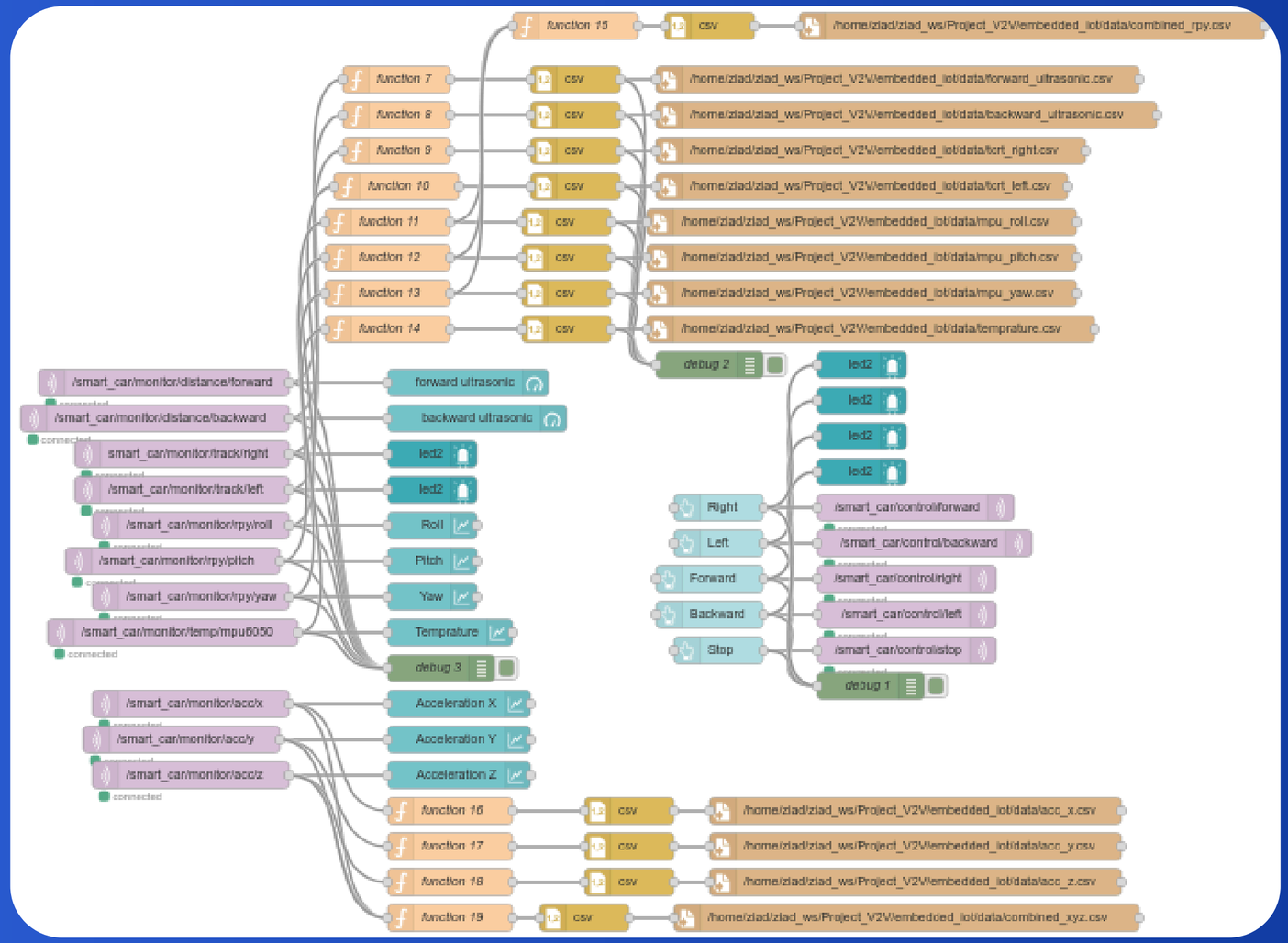
Node-RED

The IoT Visualization Layer

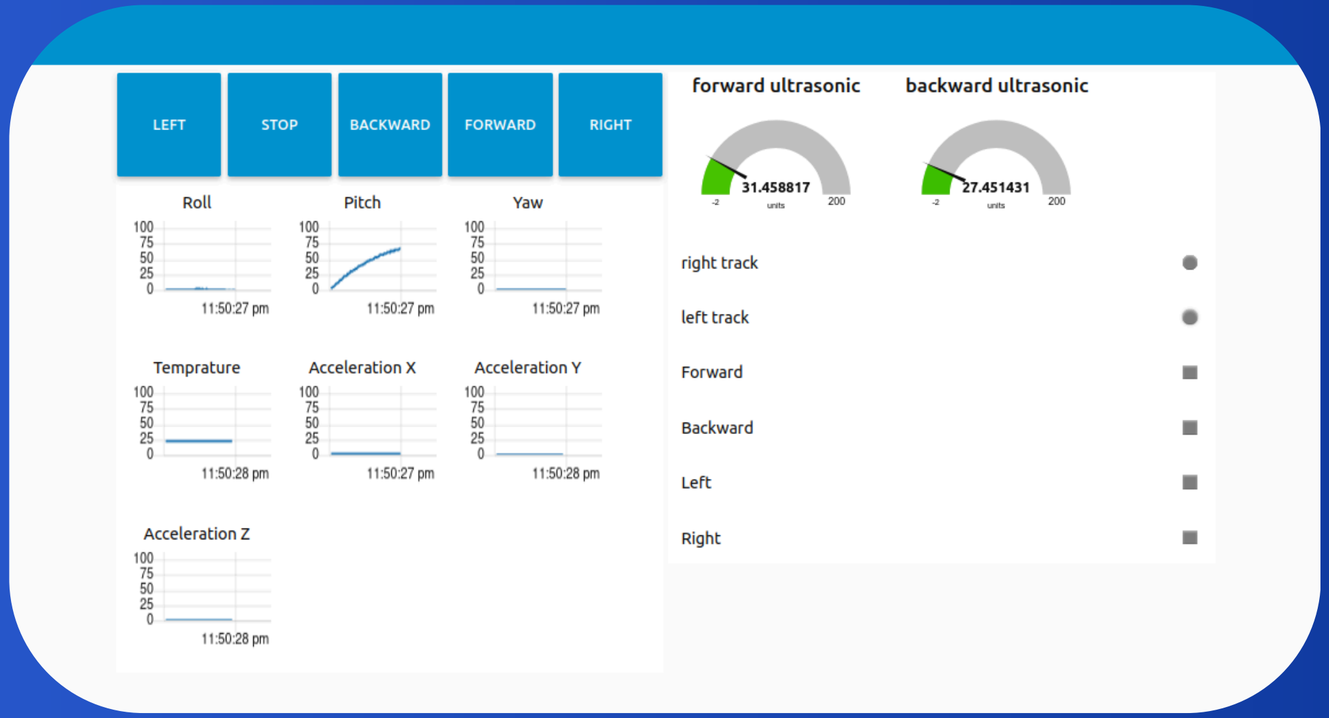
IS A POWERFUL FLOW-BASED PROGRAMMING TOOL FOR WIRING TOGETHER HARDWARE DEVICES, APIS, AND ONLINE SERVICES. IN THIS PROJECT, IT IS THE ESSENTIAL INTERFACE LAYER THAT SITS BETWEEN THE HUMAN OPERATOR AND THE MOSQUITTO MQTT BROKER.
IT ALLOWS US TO QUICKLY CREATE A GRAPHICAL WEB DASHBOARD WITHOUT WRITING EXTENSIVE FRONT-END CODE. THE FLOW YOU PROVIDED VISUALLY LINKS UI BUTTONS (LIKE "FORWARD," "BACKWARD") ON THE DASHBOARD TO MQTT OUTPUT NODES, MAPPING USER CLICKS DIRECTLY TO SPECIFIC MQTT TOPICS (/SMART_CAR/CONTROL/FORWARD). THIS ENABLES REAL-TIME REMOTE CONTROL OF THE MICRO-EXPLORER. SIMULTANEOUSLY, IT USES MQTT INPUT NODES CONNECTED TO UI LEDS TO VISUALLY MONITOR THE CURRENT STATUS OR RECEIVED COMMANDS, PROVIDING IMMEDIATE FEEDBACK AND COMPLETING THE FULL IOT CONTROL LOOP.



Node-Red Connections



Dash Board UI



Mqtt Broker

Mosquito

MOSQUITTO SERVES AS THE CENTRAL MQTT BROKER—THE MESSAGE HUB—FOR THE ENTIRE IOT SYSTEM. ITS PURPOSE IS TO RELIABLY HANDLE ALL COMMUNICATION BETWEEN THE NODE-RED DASHBOARD (THE CONTROLLER) AND THE ESP32 MICRO-EXPLORER (THE DEVICE). WHEN YOU PRESS A BUTTON ON NODE-RED (E.G., "FORWARD"), THE MESSAGE IS PUBLISHED TO MOSQUITTO. THE ESP32, WHICH IS SUBSCRIBED TO THE RELEVANT TOPIC (/SMART_CAR/CONTROL/FORWARD), INSTANTLY RECEIVES THAT MESSAGE, ENABLING REAL-TIME COMMAND AND CONTROL. IT ENSURES THE SYSTEM OPERATES ASYNCHRONOUSLY, MAKING THE COMMUNICATION ROBUST AND EFFICIENT.

Robotic Operating System ROS2 (humble)

01 Solidworks & urdf

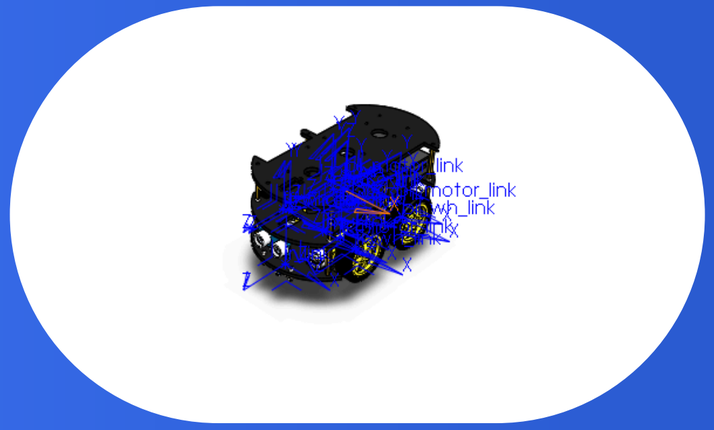
02 launch files & Rviz2

03 Gazbo Plugins

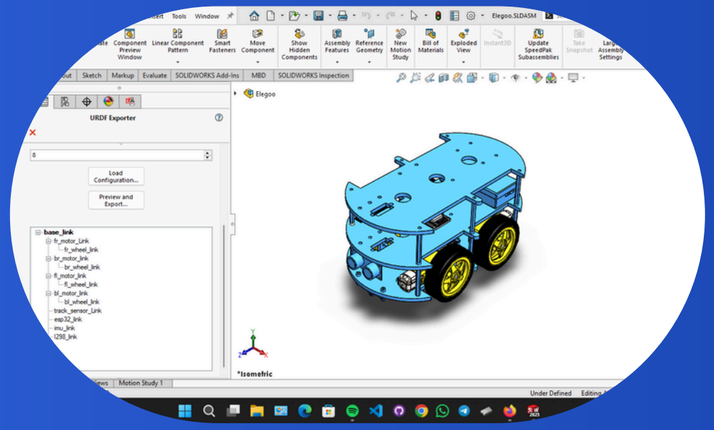
05 SLAM

06 Mapping

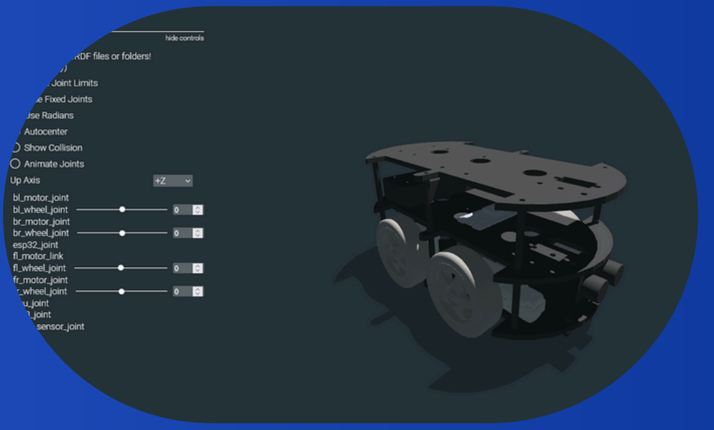
Solidworks & urdf



Define Coordinate Frames (Links)



Specify Joint Locations (Joints)

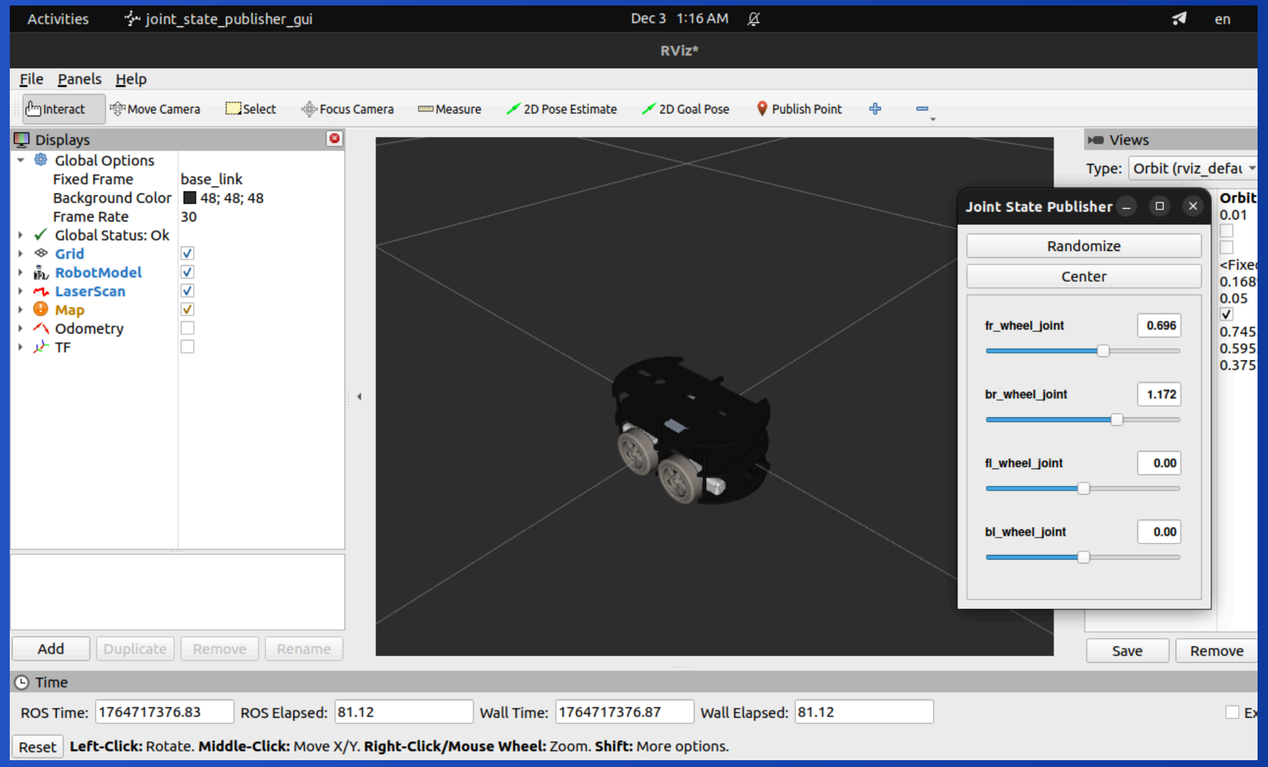


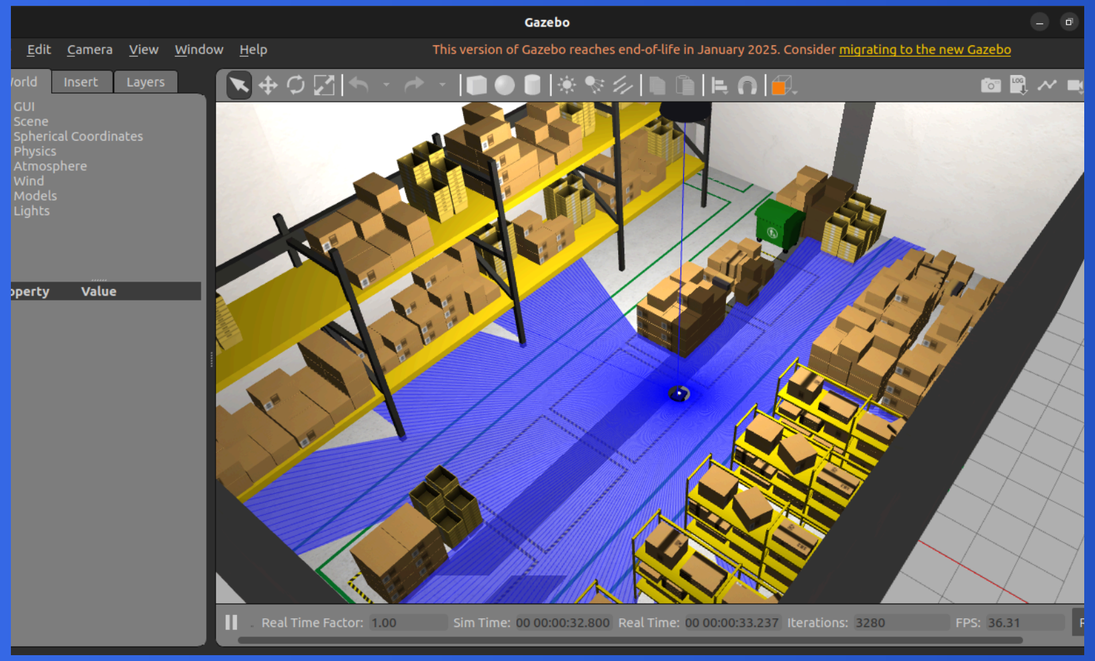
Export Mesh Files and URDF &Verify Model in URDF Viewer

Visualization

Rviz2 & Launch Files

THE LAUNCH FILE IS THE MASTER SCRIPT USED TO START ALL NECESSARY ROS 2 NODES SIMULTANEOUSLY, AUTOMATING THE ENTIRE SYSTEM INITIALIZATION. IT ENSURES THE SLAM NODE (FOR MAPPING) RUNS ALONGSIDE THE MQTT BRIDGE NODE (FOR RECEIVING CONTROL COMMANDS FROM THE ESP32) AND THE ROBOT STATE PUBLISHER (WHICH READS THE URDF). ONCE INITIALIZED, RVIZ2 IS THE CRUCIAL VISUALIZATION TOOL THAT ALLOWS US TO VIEW THE ROBOT'S 3D MODEL AND SEE THE REAL-TIME MAP BEING CONSTRUCTED VIA SLAM, PROVIDING A CRITICAL DEBUGGING AND MONITORING INTERFACE FOR THE ENTIRE PROJECT.





Simulation

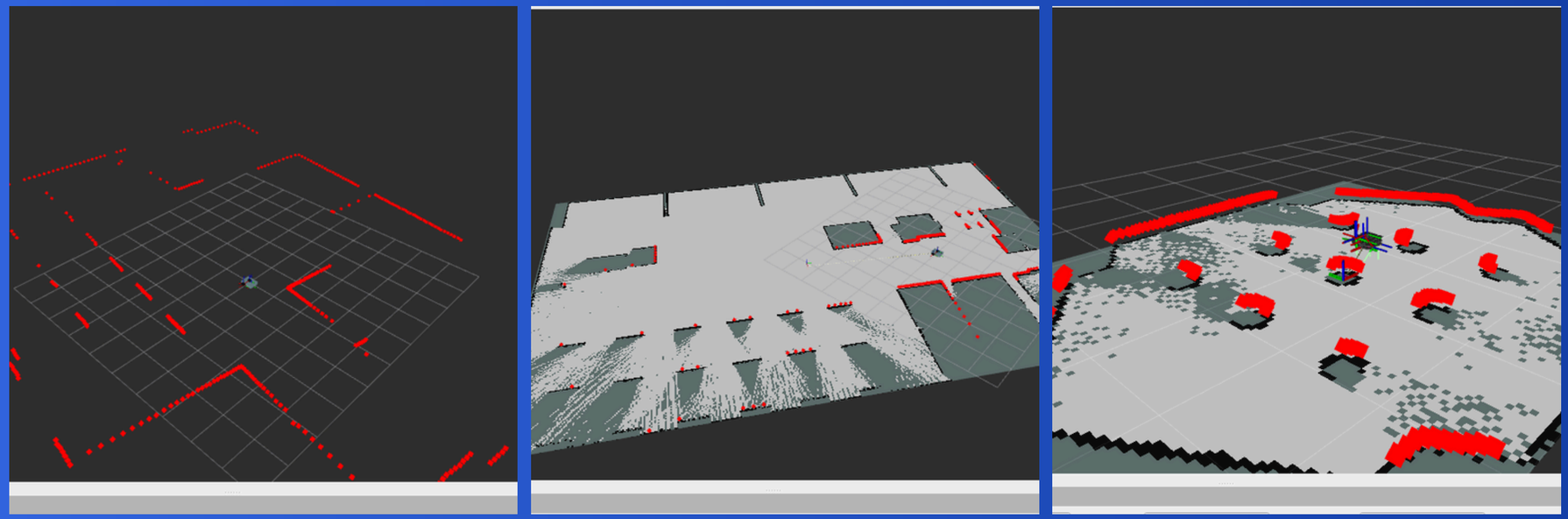
Gazebo & Launch Files

THE ENTIRE PROJECT FOCUSES ON HIGH-FIDELITY SIMULATION WITHIN GAZEBO, COMPLETELY BYPASSING REAL-WORLD HARDWARE TESTING. THE ROBOT IS PLACED WITHIN A CUSTOM-BUILT INDUSTRIAL WAREHOUSE WORLD THAT VISUALLY AND STRUCTURALLY MIMICS A REAL INDOOR ENVIRONMENT, LIKE A HOUSE OR FACTORY FLOOR, ENSURING OUR ALGORITHMS ARE ROBUSTLY TESTED AGAINST REALISTIC CONDITIONS. THIS SIMULATION IS POWERED BY THREE CORE ROS 2 PLUGINS: THE DIFFERENTIAL DRIVE PLUGIN WHICH HANDLES MOVEMENT PHYSICS AND PUBLISHES ODOMETRY DATA (/ODOM); THE RAY SENSOR PLUGIN (LIDAR), WHICH SIMULATES ENVIRONMENT SCANNING AND PUBLISHES LASERSCAN DATA (/SCAN) FOR SLAM; AND THE IMU PLUGIN, WHICH PROVIDES REALISTIC INERTIAL DATA (ACCELERATION, ANGULAR VELOCITY) TO MIMIC THE MPU-6050 AND IMPROVE POSE ESTIMATION ACCURACY. THIS SETUP CREATES A COMPREHENSIVE, SAFE, AND ENTIRELY VIRTUAL TESTING FRAMEWORK FOR DEVELOPING ADVANCED SLAM AND NAVIGATION ALGORITHMS. (500 CHARACTERS)

SLAM Tool Box

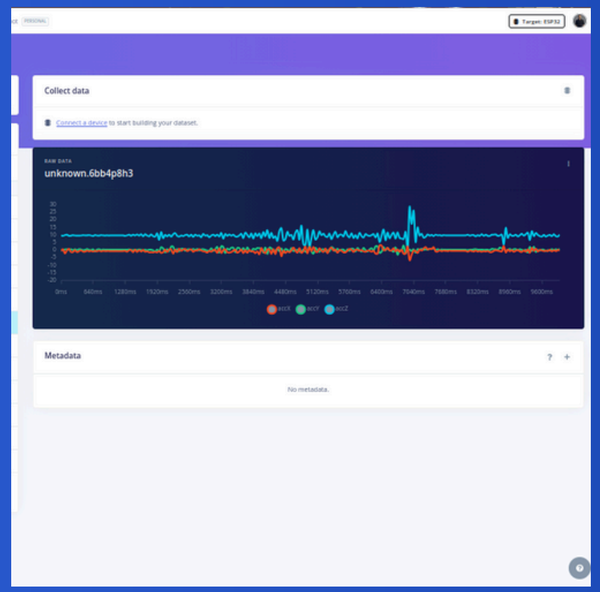
MOSQUITTO SERVES AS THE CENTRAL MQTT BROKER—THE MESSAGE HUB—FOR THE ENTIRE IOT SYSTEM. ITS PURPOSE IS TO RELIABLY HANDLE ALL COMMUNICATION BETWEEN THE NODE-RED DASHBOARD (THE CONTROLLER) AND THE ESP32 MICRO-EXPLORER (THE DEVICE). WHEN YOU PRESS A BUTTON ON NODE-RED (E.G., "FORWARD"), THE MESSAGE IS PUBLISHED TO MOSQUITTO. THE ESP32, WHICH IS SUBSCRIBED TO THE RELEVANT TOPIC (/SMART_CAR/CONTROL/FORWARD), INSTANTLY RECEIVES THAT MESSAGE, ENABLING REAL-TIME COMMAND AND CONTROL. IT ENSURES THE SYSTEM OPERATES ASYNCHRONOUSLY, MAKING THE COMMUNICATION ROBUST AND EFFICIENT.

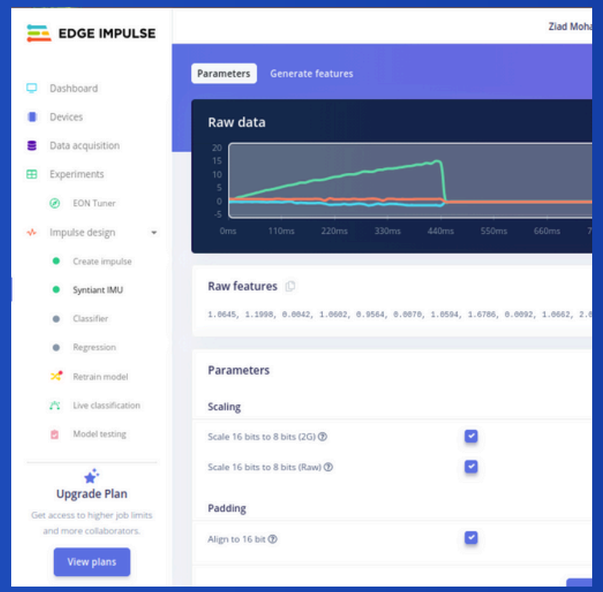
Mapping



Mapping is the process where the SLAM Toolbox uses simulated LiDAR data to build a persistent, two-dimensional occupancy grid map of the virtual industrial warehouse. This map tracks known free and occupied areas, which is fundamental for the robot's path planning and autonomous navigation.

Edge Impulse





As an optional extension, I integrated a Machine Learning (ML) layer into the project using the Edge Impulse platform. This process began by collecting raw sensor data (from components like the MPU-6050 and Ultrasonic sensors) into a CSV file, which was then uploaded for rigorous training and testing. I successfully created an Impulse capable of performing both Classification (for pattern recognition) and Regression (for continuous value prediction). This advanced ML capability significantly enhances the robot's ability to intelligently interpret sensor data, adding a powerful analytical dimension to the micro-explorer.

Ziad Fathy

2025

Thank You

SOURCE CODE