جـامـعـة الـدوحـة
للعـلـوم والـتـكـنـولـوجـيـا
UNIVERSITY OF DOHA
FOR SCIENCE & TECHNOLOGY

College of Engineering and Technology

Capstone II Project Report

Design and Implementation of an Autonomous Alignment System for Wireless Charging of Electric Vehicles

Students:

Islam Azzam, Nadine Al-Jada

Supervisor: Dr. Hassan Mahasneh

Submitted in partial fulfilment of the requirements for the degree of

The Bachelor of Science in Electrical Engineering - Automation and Control Systems Engineering (B.Sc. EE-ACSE)

The Bachelor of Science in Electrical Engineering - Electrical Power and Renewable Energy Engineering (B.Sc. EE–EPREE)

The Bachelor of Science in Electrical Engineering - Telecommunications and Network Engineering (B.Sc. EE-TNE)

DECLARATION STATEMENT

We, undersigned students, hereby declare that this project report and work described in this report is entirely our own work and has not been copied from any other source. Any material that has been used from other sources has been cited and acknowledged in proper style.

We are totally aware that any copying or improper citation of references/sources used in this report will be considered plagiarism, which is a clear violation of Academic Dishonesty Policy at the University of Doha for Science and Technology (UDST).

	Student Name	**Student ID**	**Signature**	**Date**
1	Nadine Al-Jada	60105890	Nadine	**22/11/2025**
2	**Islam Azzam**	**60105790**	Islam	**22/11/2025**

ACKNOWLEDGEMENTS

We would like to thank our supervisor, Hassan Mahasanneh, for his continuous supervision, constructive criticism, and encouragement during the preparation of this project, as well as Dr. Zeina for guiding us throughout as the instructor for both Capstone I and II.

Abstract

Wireless Electric Vehicle (EV) charging offers a safer and more convenient alternative to traditional plug-in systems, yet its adoption is limited by the need for precise alignment between the transmitter (Tx) and receiver (Rx) coils. This project addresses the challenge of enabling fully autonomous, centimeter-level alignment to make Wireless Power Transfer (WPT) practical for everyday use.

The objective is to design, build, and validate an indoor mobile robot capable of autonomously navigating, localizing, and aligning with an EV's wireless charging pad with minimal user intervention.

The methodology integrates a mecanum-wheel robotic platform, LiDAR-based SLAM for mapping, AMCL + Nav2 for navigation, encoder-based odometry, and an AprilTag camera system for final docking accuracy. A commercial WPT module is evaluated through controlled tests measuring efficiency, thermal behavior, and alignment sensitivity across varying air gaps and offsets. A web-based dashboard enables real-time monitoring, teleoperation, and visualization of maps and charging status.

Results demonstrate reliable mapping, consistent localization, accurate autonomous navigation, and successful Tx–Rx alignment within the targeted 1–2 cm tolerance. Charging tests confirm stable wireless power transfer once docking is complete.

This project showcases a complete autonomous wireless charging workflow, contributing to the advancement of smart parking systems and future autonomous EV infrastructure.

Table of Contents

List of Figures

List of Tables

Glossary of Terms / Abbreviations

[Arrange the abbreviations alphabetically]

Abbreviation	**Meaning**
AC	**Alternating Current**
ADC	**Analog-to-Digital Converter**
BMC	**Business Model Canvas**
DC	**Direct Current**
DDS	**Data Distribution Service**
DHCP	**Dynamic Host Configuration Protocol**
DWA	**Dynamic Window Approach**
EMA	**Exponential Moving Average**
EMF	**ElectroMotive Force**
EMI	**Electromagnetic Interference**
EV	**Electric Vehicles**
FoV	**Field of View**
GPIO	**General Purpose Input Output**
HF	**High Frequency**
ICNIRP	**International Commission on Non-Ionizing Radiation Protection**
IEC	**International Electrotechnical Commission**
IEEE	**Institute of Electrical and Electronics Engineers**
IPT	**Inductive Power Transfer**
I2C	**Inter-Integrated Circuit**
LiDAR	**Light Detection And Ranging**

LoS	**Line of Sight**
MOSFET	**Metal Oxide Semiconductor Field-Effect Transistor**
MQTT	**Message Queuing Telemetry Transport**
PnP	**Perspective-n-Point**
PTE	**Power Transfer Efficiency**
QoS	**Quality of Service**
ROS	**Robotic Operating systems**
RPi	**Raspberry Pi**
RSSI	**Received Signal Strength Indicator**
RT	**Real Time**
SAE	**Society of Automotive Engineers**
SDG	**Sustainable Development Goals**
SLAM	**Simultaneous Localization and Mapping**
TOF	**Time of Flight**
TPR	**Ticks Per Revolution**
UDST	**University of Doha for Science and Technology**
UI	**User Interface**
UX	**User Experience**
VHF	**Vector Field Histogram**
WPT	**Wireless Power Transfer**
YARP	**Yet Another Robot Platform**
ZVS	**Zero Voltage Switching**

1 Chapter 1: Introduction

Electric Vehicles (EVs) have, in the past years, witnessed global attention as sustainable alternatives to internal combustion engine vehicles due to their reduced emissions and environmental impact, leading to their rapid adoption [1]. However, nowadays, most EVs still rely on plug-in charging, which introduces several limitations; they are prone to wear and tear, sparking during connection, safety risks in wet or hazardous environments, and require physical effort that may be difficult for elderly users or individuals with mobility impairments. These drawbacks highlight the need for charging solutions that are safer, more convenient, and more accessible to all users [2, page 1-2].

Wireless Power Transfer (WPT), particularly Inductive Power Transfer (IPT), offers a promising solution to these challenges. In IPT systems, the Tx coil generates a high-frequency magnetic field, inducing current in the Rx coil mounted on the vehicle, thereby enabling contactless charging. Despite its advantages, IPT efficiency is strongly dependent on coil geometry, material, alignment, and space between coils, where even small misalignments can reduce efficiency significantly.

Another critical issue is that most existing wireless charging stations are static. The vehicle must be parked with high precision, often within a few centimeters, for optimal alignment between coils. Misalignment of this type can result in up to a 30% reduction in power transfer efficiency, making consistent, user-independent alignment essential for real-world deployment.

To overcome this limitation, autonomous alignment systems have emerged. Instead of requiring drivers to park with extreme precision, robotic positioning systems can automatically locate and align the charging interface. Recent advances in Simultaneous Localization and Mapping (SLAM), LiDAR-based obstacle detection, and embedded control systems now allow small robots to navigate in structured environments such as parking spaces and garages with high accuracy.

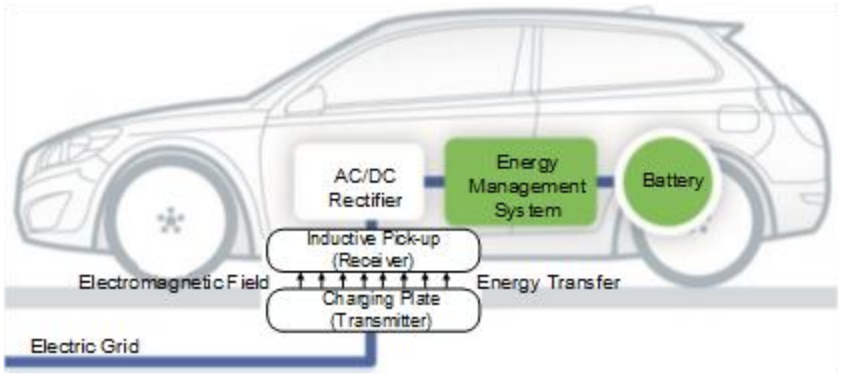


Fig. 1 Conventional Wireless EV Charging Process

1.1 Background & Motivation: (Importance of the Project, Context and Stakeholders)

As the number of EVs increases, so does the demand for charging systems that are user-friendly, reliable, and efficient. It eliminates the need for cables and manual intervention, supporting the vision of seamless and accessible charging infrastructure. However, misalignment between coils still is the primary challenge; reducing efficiency and limiting adoption. [3, page 2]

This project addresses these limitations by integrating an autonomous robot to perform precise Tx/Rx alignment, by navigating its way to EV.

The success of this project affects a varied range of stakeholders. EV owners can enjoy a charging experience that is more convenient, safe, and automated. Automotive companies and charging infrastructure suppliers can utilize this technology to provide unique and enhanced services. Urban planners in smart cities such as Doha might view this as progress toward scalable and sustainable transportation systems. Moreover, academic and research organizations can expand on this effort to drive further advancements in WPT and autonomous technologies.

1.2 Problem Statement

Existing wireless EV charging systems rely on static transmitter plates, requiring drivers to park with millimeter-level precision. Misalignment between the transmitter and receiver coils reduces efficiency by up to 30% [4], while time-consuming and prone to errors. Additionally, conventional systems lack adaptability to dynamic conditions and struggle with EMI.

Key Issues:

- Driver dependency for precise alignment with the charger.
- Inaccessibility for disabled people in traditional plug chargers.
- Power loss due to coil misalignment and gap variations.

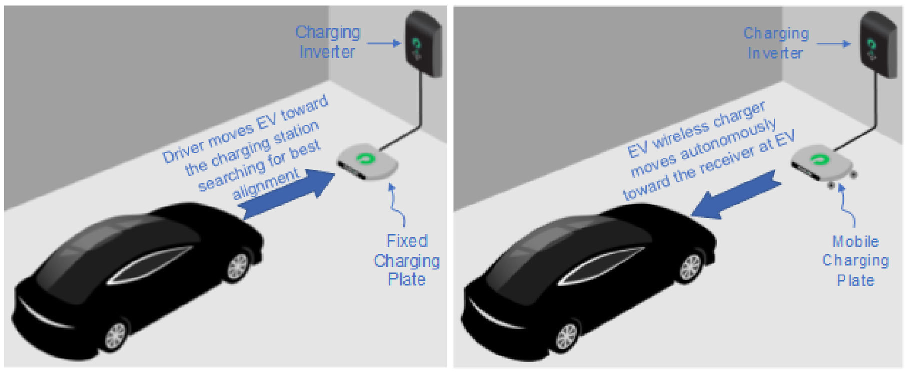

Charging Inverter
Driver moves EV toward the charging station searching for best alignment
Fixed Charging Plate
Charging Inverter
EV wireless charger moves autonomously toward the receiver at EV
Mobile Charging Plate

Fig. 2 (a) Conventional Wireless EV Charging, (b) Proposed Autonomous Mechanism.

1.3 Objectives

This project aims mainly to develop a smart and power efficient wireless charging mechanism for EV's. This can be achieved through two broad streams: wireless charger design and autonomous mechanism design.

1. Wireless charger design:

    1. Comparative analysis of WPTS technologies by evaluating each for efficiency, rating, range, safety, cost, scalability and complexity.
    2. Define technical selection criteria for the chosen charger module based on project requirements, including operating voltage, output current and transmission range; ensuring compatibility with power system and battery setup.
    3. Evaluate charging module under controlled test conditions, measuring output voltage stability, load regulation, and heat distribution across different coil alignments; to confirm safe and reliable operation.
    4. Characterize coupling efficiency by experimenting with varying air gaps (5–20 mm) and lateral offsets (0–10 mm) to quantify the power transfer loss and maintain efficiency at 80% at optimal alignment.
    5. Verify full WPT cycle (from Tx on robot chassis to Rx on test car body with a fully functional system, demonstrating safe battery charging behavior while the robot autonomously aligns its Tx with Rx.
    6. Document performance analysis comparing theoretical efficiency and measured data, identifying improvements for potential higher-power (3.7 kW) scaling in future development.

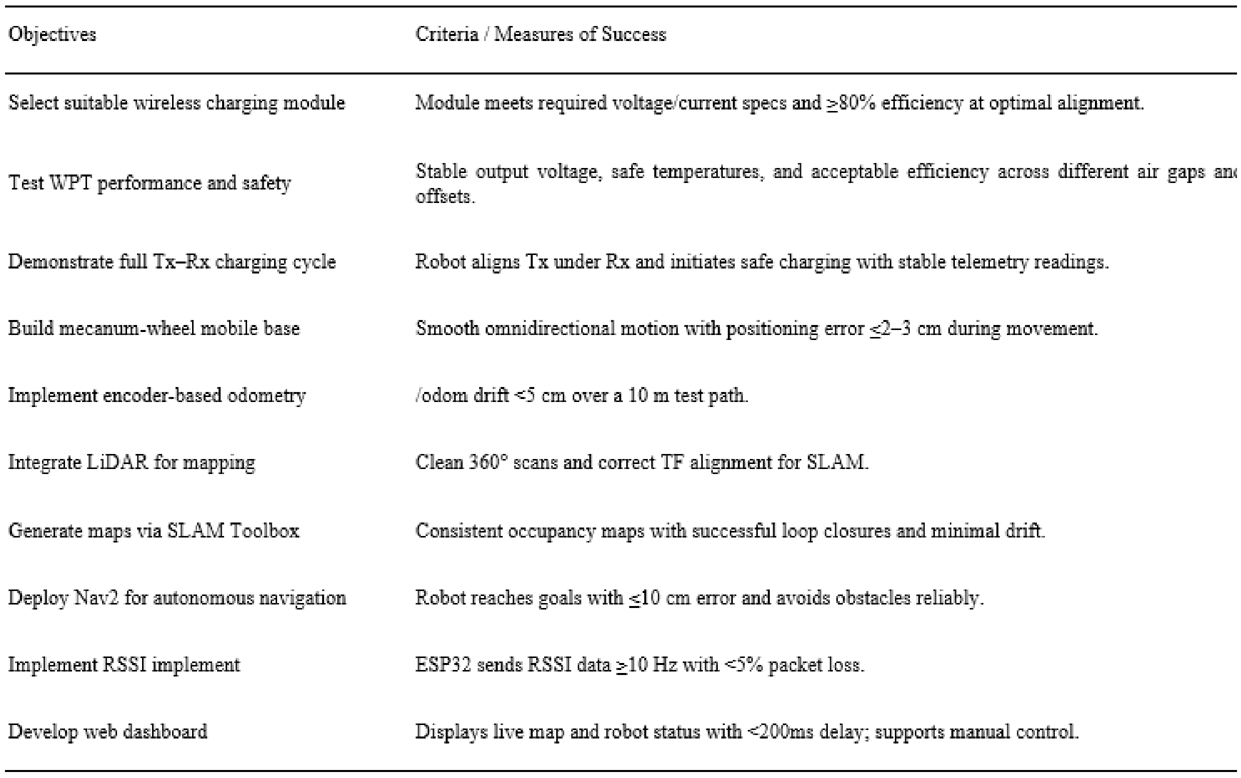
2. Autonomous Navigation and Localization System

    1. Design a 4-wheeled robotic car, equipped with mecanum wheels; to allow omnidirectional motion; to move freely and with maximum positioning accuracy
    2. Utilize DC motor encoders to provide RT wheel rotation feedback and integrate encoder data into the robot's odometry to maintain consistent pose estimation and reduce localization drift.
    3. Configure the LiDAR sensor on the robot; ensuring a clear 360° scan field and stable power supply (18 W via USB-C power bank).
    4. Implement SLAM using ROS2 Jazzy to generate RT maps of the environment and support live localization.
    5. Deploy Nav2 stack on ROS 2; to enable path planning, goal tracking, and obstacle avoidance using data from LiDAR .
    6. Set up a Wi-Fi-based communication between the RPi5 and the ESP32 localization modules to stream RT RSSI data.
    7. Develop a web-based UI to visualize live SLAM maps, monitor robot status (battery, charge level, and connection), and manually control the robot via a browser interface.

8. Optimize navigation parameters to minimize alignment time and reduce power consumption during motion.

Table 1

Measure of success



Objectives	Criteria / Measures of Success
Select suitable wireless charging module	Module meets required voltage/current specs and ≥80% efficiency at optimal alignment.
Test WPT performance and safety	Stable output voltage, safe temperatures, and acceptable efficiency across different air gaps an offsets.
Demonstrate full Tx–Rx charging cycle	Robot aligns Tx under Rx and initiates safe charging with stable telemetry readings.
Build mecanum-wheel mobile base	Smooth omnidirectional motion with positioning error ≤2–3 cm during movement.
Implement encoder-based odometry	/odom drift <5 cm over a 10 m test path.
Integrate LiDAR for mapping	Clean 360° scans and correct TF alignment for SLAM.
Generate maps via SLAM Toolbox	Consistent occupancy maps with successful loop closures and minimal drift.
Deploy Nav2 for autonomous navigation	Robot reaches goals with ≤10 cm error and avoids obstacles reliably.
Implement RSSI implement	ESP32 sends RSSI data ≥10 Hz with <5% packet loss.
Develop web dashboard	Displays live map and robot status with <200ms delay; supports manual control.

1.4 Expected Impact

The effective deployment of this smart technology is expected to change the EV ecosystem. By automating the charging, it will greatly improve user convenience and accessibility, making EV ownership more appealing and practical for a broader demographic, including the elderly and individuals with mobility challenges. From a technical perspective, the project aims to set a new benchmark for efficient and resilient WPT by mitigating alignment-related energy losses, increasing overall energy efficiency. Moreover, as a scalable and lightweight infrastructure solution, it facilitates the creation of dynamic charging services and smarter urban mobility systems. Ultimately accelerating the global transition to sustainable transportation, directly supporting environmental goals and the development of smarter, more sustainable cities.

2 Chapter 2: Literature Review

2.1 Review of Related Work: (Existing Solutions, Products, or Research, Analysis of Strengths and Gaps)

Recent advances in EV charging have increasingly focused on wireless systems that eliminate the inconvenience and mechanical wear associated with cable-based charging. Early developments in WPT concentrated on improving the efficiency of IPT by optimizing coil geometry, shielding, and resonance conditions. Research consistently shows that the coupling coefficient between Tx/Rx coils is highly sensitive to misalignment, where small offsets can significantly reduce power transfer efficiency

Researchers and industry innovators have attempted to address accessibility and convenience concerns by incorporating automation. Commercial solutions such as ZiGGY and the Car Charging Robot highlight the growing interest in mobile charging units that autonomously locate a parked vehicle and initiate charging on-demand, demonstrating the potential for hands-free charging and reduced infrastructure dependency; however, they still face limitations in alignment precision, environmental robustness, and integration with existing vehicles [5, 6]

A different method for solving the alignment problem is presented by Talaat et al. [7], where instead of moving the charging pad on the ground, they move the receiver coil installed under the vehicle. In their system, the receiver is placed on a motorized platform that shifts and tilts after parking, using computer vision and ultrasonic sensing to align with a fixed transmitter on the floor. While this approach reduces the need for precise parking, it requires additional hardware to be installed in every vehicle, increasing cost, complexity and reduces compatibility.

In contrast, our project shifts the complexity away from the vehicle and into the charging infrastructure. Instead of modifying the EV, we use a mobile robotic platform that carries the Tx coil and autonomously positions itself. The robot uses SLAM-based navigation for mapping and localization, and ultrasonic sensors for final distance alignment, making the system more flexible, easier to deploy, and suitable for existing EV models without requiring any vehicle modifications. Therefore, this

For local obstacle avoidance, one common method in mobile robotics is the DWA [8], which selects safe and feasible motion commands in real-time by evaluating the robot's possible velocities while avoiding collisions. DWA is useful for fast, reactive navigation, especially in dynamic and unpredictable environments. However, the charging environment in this projects is structured indoor space, such as a garage, requires higher localization accuracy and consistent positioning, rather than rapid reactive motion.

The project utilizes a SLAM-based navigation system (ROS2 Nav2), which generates a persistent map of the environment and continuously localizes the robot within it. This allows the robot to plan more deliberate and predictable paths. The global navigation is handled through SLAM and LiDAR mapping, while ultrasonic sensing is used during the final approach to achieve the precise alignment required between the transmitter and receiver coils. This combination provides more reliable positioning accuracy than a purely reactive method like DWA.

Selecting the appropriate software framework is essential for reliable autonomous navigation. A study by Wei et al. [9] supports the use of ROS 2, demonstrating that it provides the real-time communication, modularity, and processing efficiency required for modern robotic systems, aligning directly with our decision to implement ROS 2 Jazzy on the Raspberry Pi 5, serving as the brain of the robot.

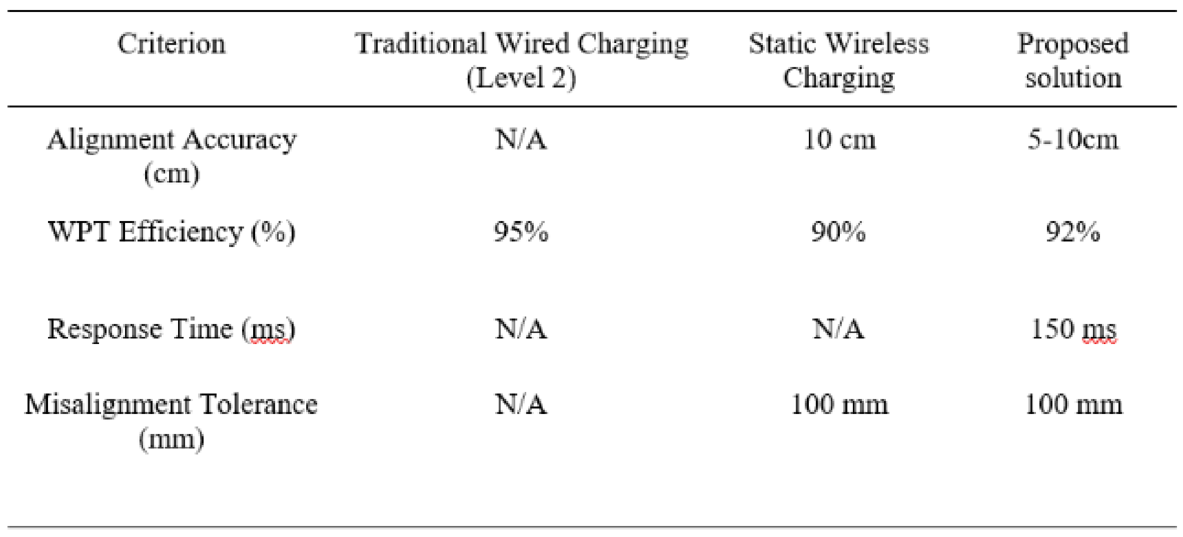
Additionally, Wei et al. highlight the effectiveness of the Nav2 and SLAM-based localization for indoor environments. Validating our choice to use LiDAR-based SLAM for mapping and localization, while relying on Nav2 for global path planning and obstacle avoidance. With this software foundation, the robot can navigate safely within a garage, maintain accurate positioning, and perform the precise alignment needed to position the Tx coil beneath the EV's Rx.

The project initially considered using RF-beacon homing to guide the robot toward the EV. However, experimental evaluation showed that RF signal strength is highly affected by multipath reflections, metallic surfaces, and environmental interference, which led to inconsistent distance estimation and insufficient alignment precision. As a result, the system transitioned to ultrasonic sensing for final alignment. Providing direct distance measurement, operate reliably in indoor environments, and offer higher precision at very low cost. When combined with SLAM for global navigation and LiDAR for obstacle detection, ultrasonic sensing enables a robust two-stage alignment strategy.

A related study demonstrated the use of ultrasonic transmitters as fixed reference beacons to estimate a mobile robot's position indoors. In that system, multiple ultrasonic sources were placed at known coordinates, and the robot measured ToF delays to determine its distance from each source. Although the report does not explicitly label the method as triangulation, the position calculation is mathematically equivalent to trilateration, where the robot's location is obtained from the intersection of multiple distance circles. The accuracy of this approach was further improved using Kalman filtering, reducing noise and achieving millimeter-level positional accuracy in controlled environments.

Table 2

Comparison of existing charger systems/ autonomous



Criterion	Traditional Wired Charging (Level 2)	Static Wireless Charging	Proposed solution
Alignment Accuracy (cm)	N/A	10 cm	5-10cm
WPT Efficiency (%)	95%	90%	92%
Response Time (ms)	N/A	N/A	150 ms
Misalignment Tolerance (mm)	N/A	100 mm	100 mm

2.2 Technical Background: (Key theories, principles, and technologies involved)

This section introduces the main concepts needed to understand an autonomous indoor mobile robot that maps, localizes, navigates, and performs precise docking for wireless EV charging. It focuses on general theory and standard technologies.

2.2.1 Robotic Software Architecture and ROS 2

Modern robots are typically built as **distributed systems** composed of small, cooperating processes rather than one monolithic program.

In the ROS 2 ecosystem:

- **Node** is an independent process that performs a specific function like reading LiDAR, computing odometry, and planning paths.
- **Topics** implement a publish/subscribe model for streaming data such as sensor readings and velocity commands.
- **Services** provide synchronous request/response, useful for operations like “save map” or “switch mode”.
- **Actions** support long-running goals with feedback and results (“navigate to pose”).
- **Parameters** hold configuration values (wheel radius, map path).

ROS 2 uses DDS underneath to handle communication with different QoS profiles for reliability and latency. It also supports lifecycle nodes, which explicitly model node states (unconfigured, inactive, active, shutting down), making system bring-up and mode switching more predictable.

This architecture encourages separation into layers such as: base control and odometry, perception and localization, planning and control, and mission management [13].

2.2.2 Coordinate Frames, Transforms, and TF2

To fuse data from multiple sensors, a robot must express all measurements in a consistent set of **coordinate frames**. Common frames for mobile robots are:

To fuse data from multiple sensors, all sensors must express their information in one language instead of their coordinate frame. To achieve this the robot converts all sensors measurements using coordinate frames and transforms

- map: a global, static frame tied to the environment.
- odom: a locally consistent frame, smooth over short time horizons.
- base_link: the robot’s body frame, typically located near its geometric center.
- Sensor frames such as laser_frame (LiDAR) or camera_link.

A transform describes how one frame relates to another (position + rotation), and the TF2 lib maintains a connected tree of transforms so the robot knows how to move data between frames. each represented as a time-stamped homogeneous transform ${}^{a}T_{b}$. Composition follows:

$$ {}^{a}T_{c} = {}^{a}T_{b}\ {}^{b}T_{c}, $$

so a point expressed in frame c can be converted into frame a by multiplying the appropriate transforms.

In typical operation:

- SLAM or AMCL publishes map → odom, anchoring odometry in the global map.
- The odometry stack publishes odom → base_link.
- robot_state_publisher uses the URDF model to publish static transforms such as base_link → wheels and base_link → laser_frame.

Tools like view_frames or Foxglove Studio visualize this TF tree, verifying that all frames are connected and transforms are broadcast at appropriate rates.

2.2.3 Mecanum Wheel Kinematics and Odometry

A **mecanum drive** uses rollers mounted at 45° around each wheel, enabling holonomic motion in the plane: the robot can move forward/backward, sideways, and rotate, often at the same time.

Let:

- v_x be forward velocity in the robot frame,
- v_y be lateral velocity (left positive),
- ω_z be yaw rate,
- r be wheel radius,
- L_x and L_y be half the wheelbase and half the track, respectively.

For an X-configuration, kinematics equation is used to compute what speed and in what direction does each wheel has to rotate to for the robot to be moving in the desired direction, given that wheels ordered front-left (1) front-right (2), rear-right (3), rear-left (4)

$$ \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} = \frac{1}{r} \begin{bmatrix} 1 & -1 & -(L_x + L_y) \\ 1 & 1 & (L_x + L_y) \\ 1 & 1 & -(L_x + L_y) \\ 1 & -1 & (L_x + L_y) \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix} $$

This equation convers desired robot motion (v_x, v_y, ω_z) into angular velocity for each wheel ω_i, these angular rates are then converted into motor commands (for example, via a scaling to “pulses per 10 ms” as expected by a motor controller).

Wheel encoders measure the cumulative rotation of each wheel in ticks. With ticks per revolution TPR, the angle increment for wheel in a time step is

$$\Delta\phi_i = 2\pi \frac{\Delta n_i}{\mathrm{TPR}},$$

where Δn_i is the tick difference. Forward kinematics combine these $\Delta\phi_i$ to compute body-frame displacement $(\Delta x_b, \Delta y_b)$ and heading change $\Delta\theta$. These increments are then rotated into a global frame and integrated over time to produce an odometric pose estimate. This odometry is smooth and locally accurate but accumulates drift due to wheel slip and unmodeled effects, so it is corrected by LiDAR-based SLAM or localization.

2.2.4 2D LiDAR Ranging and Obstacle Representation

A **2D LiDAR** emits laser pulses in a plane and measures the time until each pulse is reflected back, forming a set of ranges $r(\theta)$ for angles θ. Each scan is effectively a polar "slice" of the environment, with points converted to Cartesian coordinates in the sensor frame by

$$x = r(\theta)\cos\,\theta, y = r(\theta)\sin\,\theta.$$

LiDARs are valued for indoor robotics because they provide:

- Accurate range measurements over a wide FOV.
- Robustness to lighting changes, unlike cameras.
- Strong geometric structure (walls, corners, cars) for scan matching and localization.

For navigation, LiDAR data is typically converted into **costmaps**: grid or voxel layers where each cell holds a cost reflecting obstacle presence and distance (high cost for cells with obstacles and low cost for free space). In addition, dangerous zones around obstacles are "inflated" to keep the robot at a safe distance. [14]

2.2.5 SLAM and Occupancy Grid Mapping

SLAM solves for the robot trajectory and a map at the same time. In the 2D case, scan-matching front-ends align new LiDAR scans to either previous scans or the current map, while a back-end optimizes a pose graph to reduce drift and reconcile loop closures.

A common map representation is the occupancy grid. Space is discretized into cells, each storing the probability of being occupied. In log-odds form, the update for cell i at time t is:

$$L_t(m_i) = L_{t-1}(m_i) + \log \frac{p(m_i \mid z_t)}{1 - p(m_i \mid z_t)} - \log \frac{p(m_i)}{1 - p(m_i)},$$

By this, SLAM can add new evidence from the latest LiDAR scan, remove prior assumptions, and update the occupancy value of the cell. Loop closures, (robot returning to a position it has already visited), insert additional constraints in the pose graph and solving the resulting least-squares problem adjusts both the robot trajectory and the map, improving global consistency.

SLAM modules such as slam_toolbox can run in different modes (pure mapping, pure localization, or lifelong mapping) and typically publish both a /map topic and a TF transform map → odom [15, 16].

2.2.6 Probabilistic Localization with Particle Filters (AMCL)

When a map is already available, we use AMCL to localize, instead of SLAM. AMCL works using a particle filter, where each particle represents one possible guess of the robot's position and orientation (hypotheses of the form (x, y, θ)). At each time step, particles are propagated using an odometry-based motion model and then weighed by a measurement model that compares the expected LiDAR scan to the actual scans. Weights are proportional to the likelihood $p(z_t \mid x_t)$, and resampling concentrates particles where the match is best (particles with the highest weight are decided to be the best estimate, or their average if many). Formally for particle k:

$$w_t^{(k)} \propto w_{t-1}^{(k)}\, p(z_t \mid x_t^{(k)}),$$

followed by resampling to obtain a new set $\{x_t^{(k)}\}$. AMCL typically outputs this via the TF chain as a map → odom transform, which, combined with odom → base_link, yields a full pose of the robot in the map frame [18,20].

2.2.7 Global Planning, Costmaps, and Controllers (Nav2)

Robot navigation in a static or semi-static environment is usually decomposed into:

- **Global planning**, which finds a path through a static map from the current pose to a target pose, often using graph search (Dijkstra, A*, NavFn).
- **Local planning**, which generates short-horizon feasible trajectories considering dynamics and close obstacles.
- **Feedback control**, which converts local plan segments into velocity commands while respecting velocity and acceleration limits.

Costmaps integrate static obstacles (from the occupancy grid map) with dynamic obstacles (from LiDAR or other sensors). They store costs that rise as the robot approaches walls or objects; planners search for paths that minimize the integral of these costs.

Nav2 is built around a plugin architecture, meaning both the planners and controllers can be swapped or customized depending on the robot's needs. A local planner such as DWA (Dynamic Window Approach) samples candidate velocities (v_x, v_y, ω_z), simulates them over a short time. Every simulated trajectory is scored using cost functions:

a. Distance to path (how well it follows the planned route)
b. Distance to goal (how much progress it makes)
c. Proximity to obstacles (based on the local costmap)

and chooses the lowest-cost option. Behavior Trees orchestrate high-level navigation logic: computing a path, following it, reacting to new goals, and executing recovery actions (clearing costmaps, spinning, backing up) when progress stalls [10,17, 19, 21].

2.2.8 Camera Calibration and Fiducial Pose Estimation

An RGB camera can be modeled by the pinhole camera equation:

$$s\begin{bmatrix}u\\v\\1\end{bmatrix} = K[R|t]\begin{bmatrix}X\\Y\\Z\\1\end{bmatrix}$$

where (X, Y, Z) are 3D coordinates in the camera frame, (u, v) are pixel coordinates, K is the intrinsic matrix (focal lengths and principal point), and (R, t) is the camera pose. Real lenses introduce distortion, modeled by parameters estimated during calibration.

For fiducial markers such as April-Tags or ArUco codes, the marker has known geometry (square of known size). Detecting the corners obtains 2D–3D correspondences of image. Solving the PnP problem provides the rotation and translation [R|t] that best align the known marker points in 3D to the observed 2D projections, minimizing reprojection error.[25]

The resulting translation vector $\mathbf{t} = (x, y, z)^\top$ gives the marker position relative to the camera. This can be used for simple visual servoing: for example, to maintain a desired standoff distance z^* and lateral alignment $x^* = 0$,

$$v = -k_v(z - z^*), \omega = -k_\omega x,$$

where v is a forward velocity, ω is a yaw rate, and k_v, k_ω are gains. These commands can be mapped into `cmd_vel` to refine robot pose near a docking station. [24]

2.2.9 ESP-NOW

ESPs conventionally used IEEE 802.11 for data transmission, meaning devices were required to connect to a Wi-Fi AP, obtain an IP address using DHCP, then establish TCP/ UDP connections for data exchange. While effective, it introduces latency, packet overhead, and increased power consumption, thus, impractical for small-scale projects

For simple systems that only need to exchange little data locally, the full Wi-Fi and IP stack is unnecessarily complex and energy-intensive. This led Espressif developers to create ESP-NOW to allow direct, low-power, and connectionless communication between ESP devices without the need for routers or internet infrastructure. By using data-link layer technology to operate below the IP layer, bypassing network authentication, DHCP configuration, and socket management, solving the power and complexity issues related to WiFi [22]

ESP-NOW uses 2.4 GHz similar to standard Wi-Fi IEEE 802.11 networks. However, instead of using Wi-Fi data frames for communication, ESP-NOW encapsulates its payload within 802.11 action frames, allowing ESP devices to communicate directly at the hardware level. Because it operates on the same band and transceiver hardware, it can coexist with Wi-Fi communication or even function in hybrid modes alongside Wi-Fi, bridging the gap between Bluetooth (simple but limited range/ bandwidth) and Wi-Fi (high throughput but high overhead).

2.2.10 Wireless Power Transfer, Coupling, and Misalignment (Overview)

Near-field inductive WPT is based on Faraday's law: a time-varying magnetic field produced by a primary coil induces an EMF in a secondary coil (or a nearby conductor) $\varepsilon = db/dt$. In most EV applications, both coils are tuned to resonate at the same frequency to maximize power transfer and efficiency.

The effective coupling is expressed by the coupling coefficient $k \in [0,1]$ and the quality factors Q_1, Q_2 of the coils. Higher k and Q values generally lead to higher link efficiency. Lateral, vertical, or angular misalignment reduces mutual inductance $M = k\sqrt{L_1 L_2}$, directly lowering the power delivered for a given primary current. Standard-compliant systems therefore specify tolerances on allowable misalignment; meeting those tolerances drives the need for accurate and repeatable pad positioning. [23]

3 Chapter 3: Project Requirements and Specifications

This chapter defines the essential requirements, constraints, and considerations necessary for the project's successful implementation. It ensures clarity in system design, functionality, and compliance with standards.

3.1 Customer / Stakeholder Needs

The proposed autonomous wireless EV charging robot is designed for malls, public parking, and car companies by being affordable, reliable, and easy to operate. It's fully autonomous; requiring no user interaction, improving convenience and uncongested traffic in busy facilities. It is safe; using LiDAR, obstacle avoidance, and controlled motion for operation around vehicles and pedestrians. The system is also compatible with different layouts and floor types, making it suitable for the real world. Hardware/ software are modular and easy to run and maintain, with documentation explaining updates and repairs. For car companies, it demonstrates an innovative, scalable solution that supports the transition toward smart, automated, and sustainable EV charging.

3.2 Requirements

3.2.1 User Experience Considerations

The user experience section explains how parking facilities, mall operators, and EV owners interact with the autonomous system and what expectations they have.
Our design focuses on simplicity, reliability, and low-effort operation.

Table 3

UX ASPECT

UX Aspect	Implementation of an autonomous wireless charging system
Comfort	Robots operate quietly and smoothly, and alignment with EV is automatic.
Portability	System uses compact, commercially available, hardware, installation made easy in different parking layouts; no large infrastructure.
Ease of use	Entirely autonomous + simple dashboard for monitoring maps, robot location, and charging status IRT.
safety	LiDAR-based obstacle detection

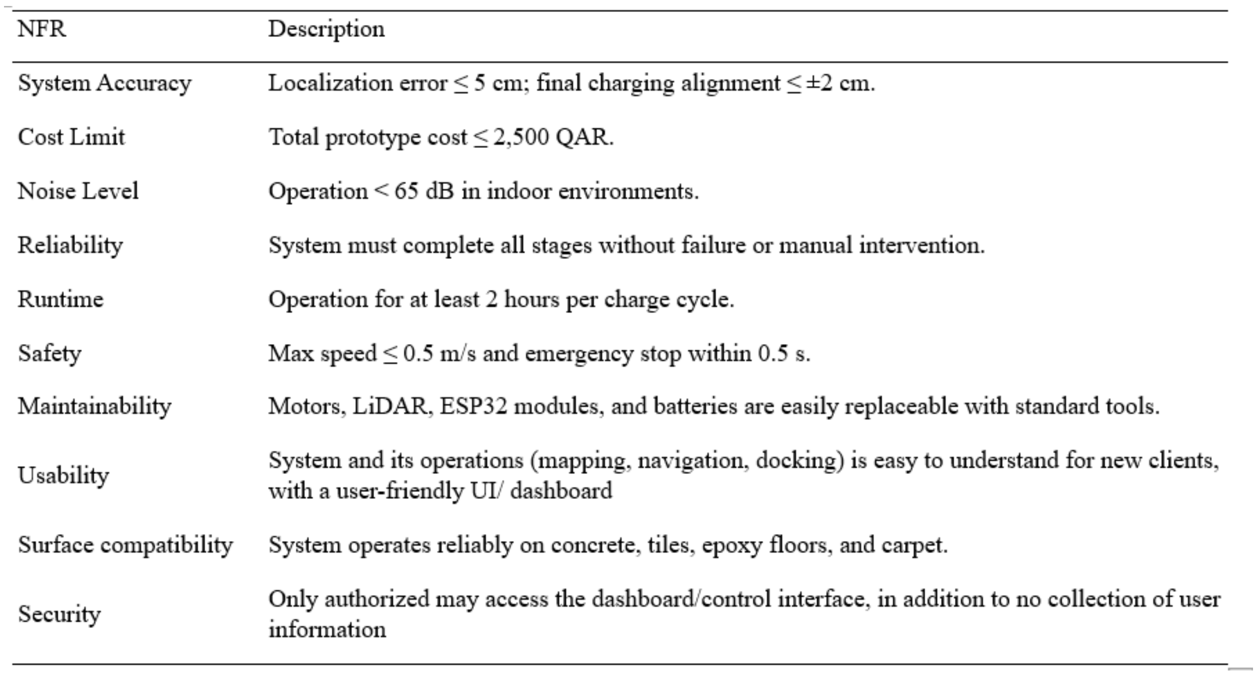
3.2.2 *Functional Requirements*

Table 4
Functional Requirement

FR	Expected behavior
Navigation	The robot must autonomously navigate a space using SLAM-based mapping and localization and reliably reach target position.
Obstacle avoidance	The robot must detect static/ dynamic obstacles using its LiDAR sensor and avoid collisions IRT while maintaining a safe and smooth path.
Final alignment	The robot must perform precise final alignment to position Tx coil on Rx coil with ≤ 2 cm lateral error, using camera
Motion control	The robot must control all motors using encoder feedback while keeping the ride stable, smooth, and accurate motion in all directions with consistent odometry updates.
Manual override	The system must provide a UI to allow the user to override autonomous behavior and drive the robot manually when needed.
Charging feedback	The system must showcase when wireless charging has successfully started using UI
Maps	The robot must be able to create, save, and load maps for navigation across different floors/ environments. For uploaded maps, the robot must accurately localize itself using AMCL without requiring remapping each time.
RT telemetry monitoring	The system must always display RT telemetry on dashboard
Wireless communication	The robot must maintain wireless communication with the dashboard and vehicle-side charger to exchange control and data.

3.2.3 Non-Functional Requirements

Table 5
Non-functional requirement



NFR	Description
System Accuracy	Localization error ≤ 5 cm; final charging alignment ≤ ±2 cm.
Cost Limit	Total prototype cost ≤ 2,500 QAR.
Noise Level	Operation < 65 dB in indoor environments.
Reliability	System must complete all stages without failure or manual intervention.
Runtime	Operation for at least 2 hours per charge cycle.
Safety	Max speed ≤ 0.5 m/s and emergency stop within 0.5 s.
Maintainability	Motors, LiDAR, ESP32 modules, and batteries are easily replaceable with standard tools.
Usability	System and its operations (mapping, navigation, docking) is easy to understand for new clients, with a user-friendly UI/ dashboard
Surface compatibility	System operates reliably on concrete, tiles, epoxy floors, and carpet.
Security	Only authorized may access the dashboard/control interface, in addition to no collection of user information

3.3 Design Constraints (Cost, Power, Voltage, Speed, Safety, Size, Environmental Factors)

This project involves several constraints that significantly influenced the approaches chosen and the timeline of completion, like:

1. Time constraint: the development and completion is limited to 2 academic semesters divided equally for research/ development and implementation.

2. Budget constraint: Simple, cost-effective, accurate components and testing softwares are prioritized to keep cost under 2000Qar.

3. Component availability constraint: Most of components needed such as a rigid metal car body with odometry motor drivers, Tx/Rx modules were unavailable locally, and must be ordered online.

4. Technical constraint: Technical challenges in reaching the desired efficiency and accuracy in various environments (non-ideal conditions) and validating results through testing.

5. Testing and experimental constraints: Testing is only confined to university labs and empty parking spaces, as real-world validation is restricted due to the lack of full-scale EV integration.

3.4 Ethical, Safety, and Environmental Considerations

1. Ethical considerations: No collection of user data beyond charging status, as the app uses anonymized MQTT topics all the data is shared locally within the user network to ensure the user privacy. All third-party code follows open-source licensing (ESP-IDF, ROS2).

2. Safety consideration: Electrical safety includes using low-voltage components for testing and insulated wiring to prevent shocks, in addition to LIDAR-based collision-avoidance algorithms with response time < 200 ms, supported by a web app that will notify the user of any emergencies, and RT monitoring.

3. Environmental consideration: All ESP32 modules operate at < 100 mW within IEEE C95.1 human exposure limits, LiDAR, motors, and compute nodes are optimized to minimize unnecessary CPU load, motor current, and idle power. Structural components are reusable, and electronic waste is minimized with no harmful emission.

3.5 Standards and Regulations (IEEE, IEC, etc.)

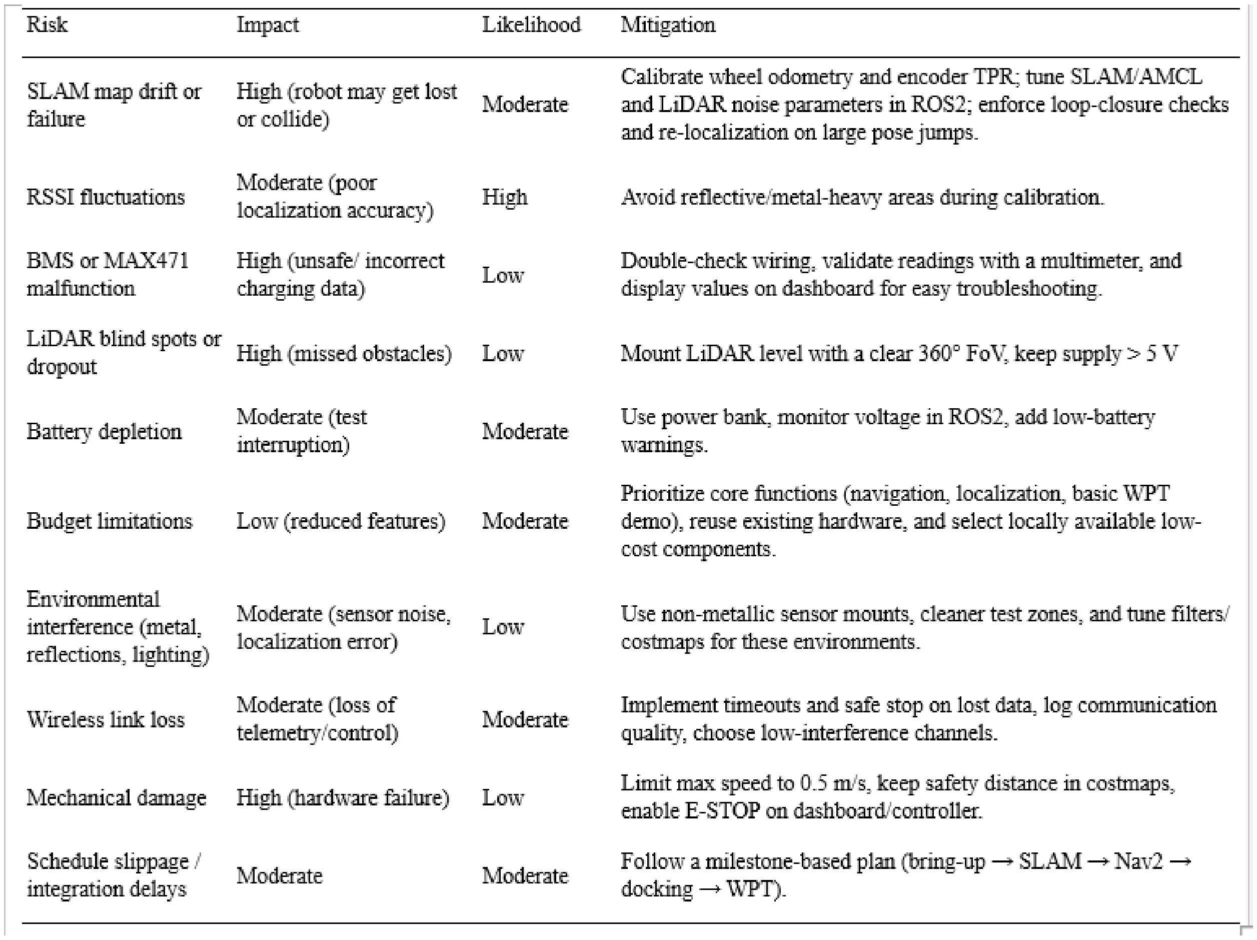
Table 6
Standards and regulations

Standard / Regulation	Applies To	Description
SAE J1772	On-board chargers	Level 1 AC: 120V_{AC}, Power 0-1.9kW
		Lever 2 AC: 240V_{AC}, Power 1.9-19.2kW
IEC 61980	EV WPTS	Ensure efficiency, electrical safety, and compatibility with grid infrastructure
IEC 61010	Electrical Safety	Safe design for low-voltage electronic devices
ROS2 Open-Source License (Apache 2.0)	Software framework	Defines usage and redistribution rights
IEEE 829 / ISO 29119	Software Test Documentation	Applied to firmware testing and validation

Note: Sourced from [27-31]

3.6 Risk Assessment and Mitigation

Table 7
Risk assessment and mitigation



Risk	Impact	Likelihood	Mitigation
SLAM map drift or failure	High (robot may get lost or collide)	Moderate	Calibrate wheel odometry and encoder TPR; tune SLAM/AMCL and LiDAR noise parameters in ROS2; enforce loop-closure checks and re-localization on large pose jumps.
RSSI fluctuations	Moderate (poor localization accuracy)	High	Avoid reflective/metal-heavy areas during calibration.
BMS or MAX471 malfunction	High (unsafe/ incorrect charging data)	Low	Double-check wiring, validate readings with a multimeter, and display values on dashboard for easy troubleshooting.
LiDAR blind spots or dropout	High (missed obstacles)	Low	Mount LiDAR level with a clear 360° FoV, keep supply > 5 V
Battery depletion	Moderate (test interruption)	Moderate	Use power bank, monitor voltage in ROS2, add low-battery warnings.
Budget limitations	Low (reduced features)	Moderate	Prioritize core functions (navigation, localization, basic WPT demo), reuse existing hardware, and select locally available low-cost components.
Environmental interference (metal, reflections, lighting)	Moderate (sensor noise, localization error)	Low	Use non-metallic sensor mounts, cleaner test zones, and tune filters/ costmaps for these environments.
Wireless link loss	Moderate (loss of telemetry/control)	Moderate	Implement timeouts and safe stop on lost data, log communication quality, choose low-interference channels.
Mechanical damage	High (hardware failure)	Low	Limit max speed to 0.5 m/s, keep safety distance in costmaps, enable E-STOP on dashboard/controller.
Schedule slippage / integration delays	Moderate	Moderate	Follow a milestone-based plan (bring-up → SLAM → Nav2 → docking → WPT).

3.7 Sustainable Development Goals (SDGs) Mapping

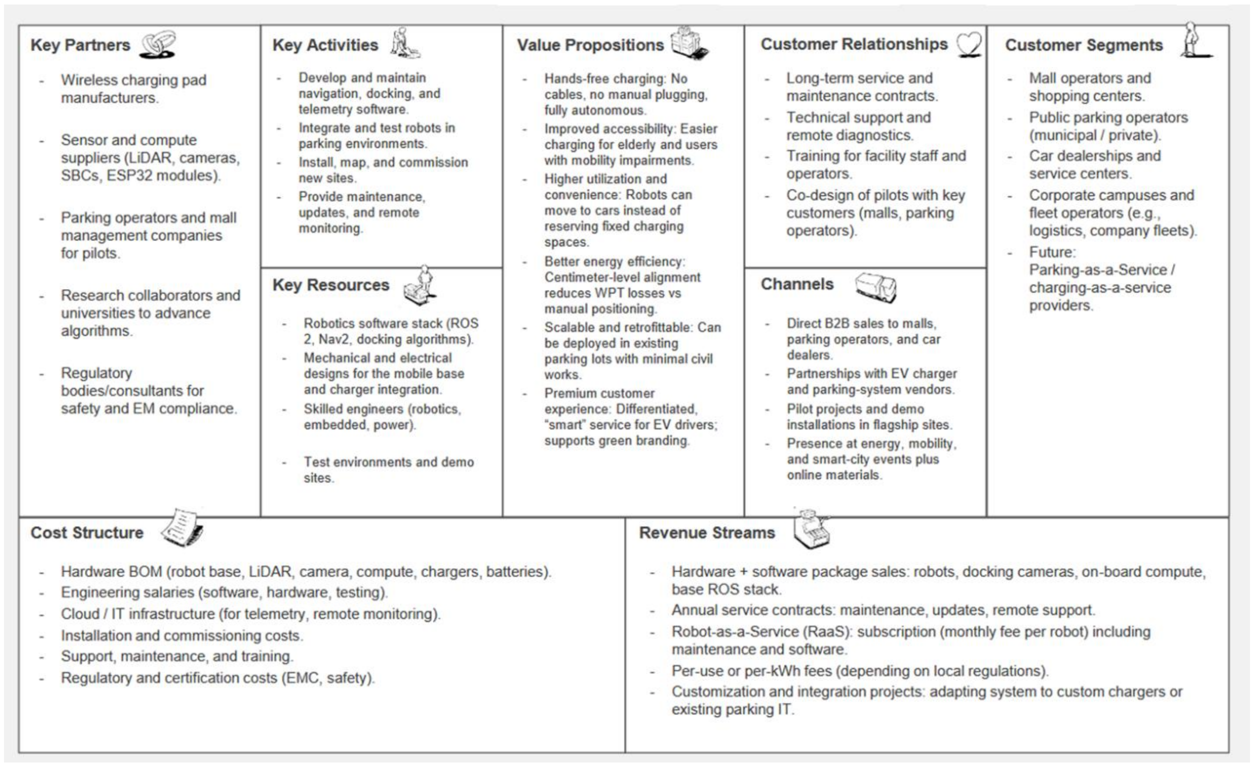
Table 8
SDG mapping

SDG	How project contributes to SDG
SDG 7 - Affordable and Clean Energy	Promotes the adoption of EVs with auto-alingment, making transportation more practical, accessible, and affordable for everyday users.
SDG 9 - Industry, Innovation, and Infrastructure	Encourages technological innovation in Qatar’s transportation by integrating advanced localization and WPT technologies.
SDG 11 - Sustainable Cities and Communities	Improves urban mobility and sustainability by automating the EV charging process, reducing the need for parking space and manual plug-in operations as the system is compact, smart, sustainable and aligns with Qatar’s national vision.
SDG 13 - Reduce greenhouse gas emissions	Helps reduce greenhouse gas emissions by encouraging EV adoption over internal combustion vehicles. WPT convenience eliminates user barriers like cable handling, promoting cleaner transport. Estimated 30–50% CO_2 reduction potential depending on renewable energy share in the grid.

Note: taken from [26]

6. 3.8 Economic evaluation, Business Model Canvas (BMC)

The economic viability of the autonomous wireless EV charging robot has been analyzed using a Business Model Canvas (BMC), summarized in Fig. 3 below. The BMC highlights malls, public parking operators, and car dealerships as the primary customer segments, reflecting the project’s focus on high-turnover, indoor parking environments. The core value proposition is hands-free, autonomous charging that improves accessibility, reduces driver effort, and increases the effective utilization of existing parking and charging infrastructure. Revenue is expected from hardware and software package sales, long-term service and maintenance contracts, and potential Robot-as-a-Service (RaaS) models. Key activities and resources include ongoing development of the ROS 2–based navigation and docking stack, system integration, and site commissioning, supported by partnerships with wireless charger manufacturers and parking-system vendors. Overall, the BMC indicates that, beyond its technical feasibility, the proposed system can fit into a realistic, service-oriented business model for smart parking facilities.


Key Partners

- Wireless charging pad manufacturers.
- Sensor and compute suppliers (LiDAR, cameras, SBCs, ESP32 modules).
- Parking operators and mall management companies for pilots.
- Research collaborators and universities to advance algorithms.
- Regulatory bodies/consultants for safety and EM compliance.

Key Activities

- Develop and maintain navigation, docking, and telemetry software.
- Integrate and test robots in parking environments.
- Install, map, and commission new sites.
- Provide maintenance, updates, and remote monitoring.

Key Resources

- Robotics software stack (ROS 2, Nav2, docking algorithms).
- Mechanical and electrical designs for the mobile base and charger integration.
- Skilled engineers (robotics, embedded, power).
- Test environments and demo sites.

Value Propositions

- Hands-free charging: No cables, no manual plugging, fully autonomous.
- Improved accessibility: Easier charging for elderly and users with mobility impairments.
- Higher utilization and convenience: Robots can move to cars instead of reserving fixed charging spaces.
- Better energy efficiency: Centimeter-level alignment reduces WPT losses vs manual positioning.
- Scalable and retrofittable: Can be deployed in existing parking lots with minimal civil works.
- Premium customer experience: Differentiated, "smart" service for EV drivers; supports green branding.

Customer Relationships

- Long-term service and maintenance contracts.
- Technical support and remote diagnostics.
- Training for facility staff and operators.
- Co-design of pilots with key customers (malls, parking operators).

Channels

- Direct B2B sales to malls, parking operators, and car dealers.
- Partnerships with EV charger and parking-system vendors.
- Pilot projects and demo installations in flagship sites.
- Presence at energy, mobility, and smart-city events plus online materials.

Customer Segments

- Mall operators and shopping centers.
- Public parking operators (municipal / private).
- Car dealerships and service centers.
- Corporate campuses and fleet operators (e.g., logistics, company fleets).
- Future: Parking-as-a-Service / charging-as-a-service providers.

Cost Structure

- Hardware BOM (robot base, LiDAR, camera, compute, chargers, batteries).
- Engineering salaries (software, hardware, testing).
- Cloud / IT infrastructure (for telemetry, remote monitoring).
- Installation and commissioning costs.
- Support, maintenance, and training.
- Regulatory and certification costs (EMC, safety).

Revenue Streams

- Hardware + software package sales: robots, docking cameras, on-board compute, base ROS stack.
- Annual service contracts: maintenance, updates, remote support.
- Robot-as-a-Service (RaaS): subscription (monthly fee per robot) including maintenance and software.
- Per-use or per-kWh fees (depending on local regulations).
- Customization and integration projects: adapting system to custom chargers or existing parking IT.

Fig. 3 BMC

4 Chapter 4: Methodology and Design Approach

4.1 Introduction to Methodology

This project focuses on designing, implementing, and testing an autonomous robot capable of navigating, localizing, and aligning with an EV's wireless charging pad. The methodology follows a layered, system-level approach, where each layer builds on the one below it for reliable workflow.

At the lowest layer, we included mecanum-wheel base, allowing us to implement mecanum kinematics, interfacing with a four-channel motor/encoder driver over I2C, and using wheel encoders to generate accurate odometry and joint states that match the robot's URDF model. A consistent TF2 tree is maintained (map → odom → base_link → sensors) to ensure all sensors share a common reference.

Above that, the navigation layer includes a LiDAR-centric perception and navigation layer is built. The YDLIDAR provides 2D scan data, fed into slam_toolbox for mapping mode and into AMCL + Nav2 for navigation mode. Using these tools, the robot can build accurate occupancy-grid maps and later localize itself within them to autonomously move through indoor environments.

On top of navigation, the mission-management layer handles semantic information such as named locations, stored per map using custom ROS2 services. The state_manager node coordinates all high-level operations, such as switching between mapping and navigation, saving/loading maps, tracking the current robot pose, and broadcasting unified

/robot_state information. In addition, it supervises SLAM and Nav2 as separate lifecycle processes, ensuring that only the correct one is running at a time.

After that, a camera-based docking layer is added for further accuracy. A calibrated camera detects a QR code placed near each charging target. The estimated tag pose is converted into small linear and angular corrections, allowing the robot to fine-tune its position after Nav2 navigation. Simultaneously, a small ESP32-based telemetry link sends RT voltage/ current readings, and charging data from Rx side to the robot.

By this, we have ensured that each subsystem is implemented, tested, and validated independently, then fused into a complete autonomous wireless charging system.

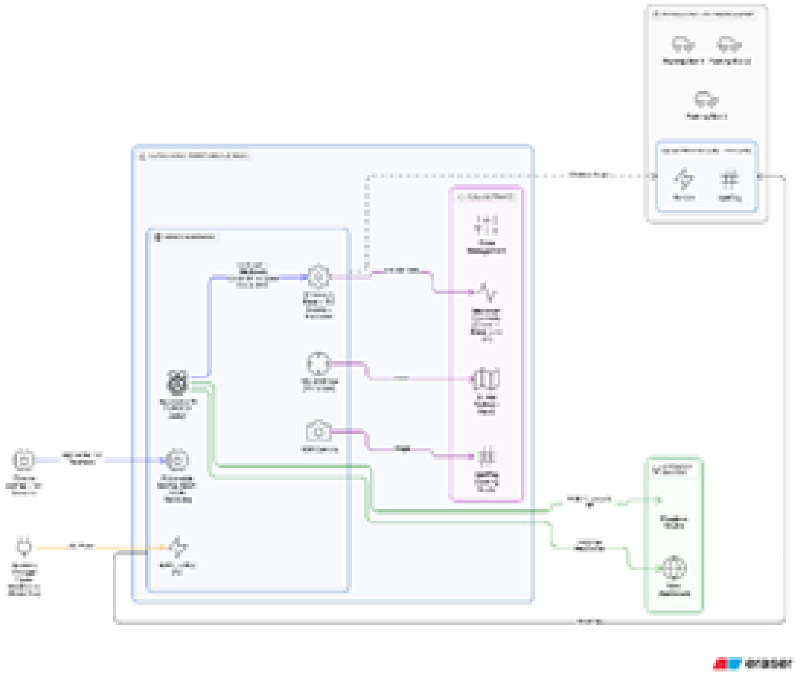


Fig. 4 full architecture of autonomous system

4.2 System Architecture (Block Diagrams, Functional Flow)

The system is organized as a layered architecture running on a RPi 5, written largely in ROS2 Jazzy, with a browser-based dashboard and a small ESP-NOW telemetry link to the charger.

At the physical level the metal base chassis with four DC motors and mecanum wheels are used as well as a 2D LiDAR derived by the ydlidar_ros2_driver and its C++ SDK, an RGB camera mounted on the front/top, and a commercial wireless charging pad mounted on the front of the robot (under the camera). The EV carries the Rx pad and a small esp32 module connected to a voltage and current sensors on the charger's DC output.

At the ROS 2 layer, the code is structured into several packages. "interfaces" package holds custom service definitions such as SetMode, SaveMap, SaveParkingSpot, and other that are used to define the shape of the messages used throughout the communication between the different ROS 2 nodes. Motor packages implements all low-level drive control and odometry, Including the I2C motor driver interface, mecanum kinematics, encoder processing, and conversion of encoders to joint states. The “robot” package contains higher-level nodes such as state_manager, and mode_manager, URDF description and TF publishing, and integration launch files. The “camera” package contains the AprilTag based docking node and camera calibration and tag generator helping scripts. Finally third-party packages including ydlidar_ros2_driver for the LiDAR, slam_toolbox for SLAM, nav2 and nav2_bringup for the navigation stack, and rosbridge_server for WebSocket access from the dashboard.

The top level launch is handled by a script (bringup_all) this launch file starts:

- The robot state publish and joint state publisher with the URDF from the defined xml file. This defines the static geometry and TF of the robot (including wheels and laser frame).
- The motor controller for robot control, odometry node for odometry calculations, and joystick controller node to allow manual joystick control through a controller such as the PlayStation 5 controllers.
- The lidar driver using their package, which published the LaserScan messages on the /scan topic to be used in the SLAM and Nav2.

A separate integration launch file starts:

- The mode manager node manages the lifecycle of the SLAM toolbox and Nav2 as external processes.
- The state manager node provides a high level service for robot status and publishing /robot_state to be used by the dashboard and any other optional nodes.
- Rosbridge server to start the WebSocket server so that the dashboard can communicate with our ros2 system.
- The Node.js web server serves the HTML/CSS/JS front-end and points it to our WebSocket server.

In mapping mode, the mode manager launches slam toolbox in mapping mode with custom configuration set in a predefined file. In navigation mode, it shuts down SLAM and launches the Nav2 localization (to serve the map so other nodes can view it and manages the robot localization in the map) and the navigation bringup using a custom predefined configuration to manage the be used for the robot goal assignment and obstacle avoidance.

The state manager continuously monitors the Nav2 action server, the TF tree, and the map directory, and publishes a consolidated /robot_state JSON string summarizing the current mode, map name, Nav2 readiness, and robot pose. The dashboard reads this to update UI and to refresh parking spot lists.

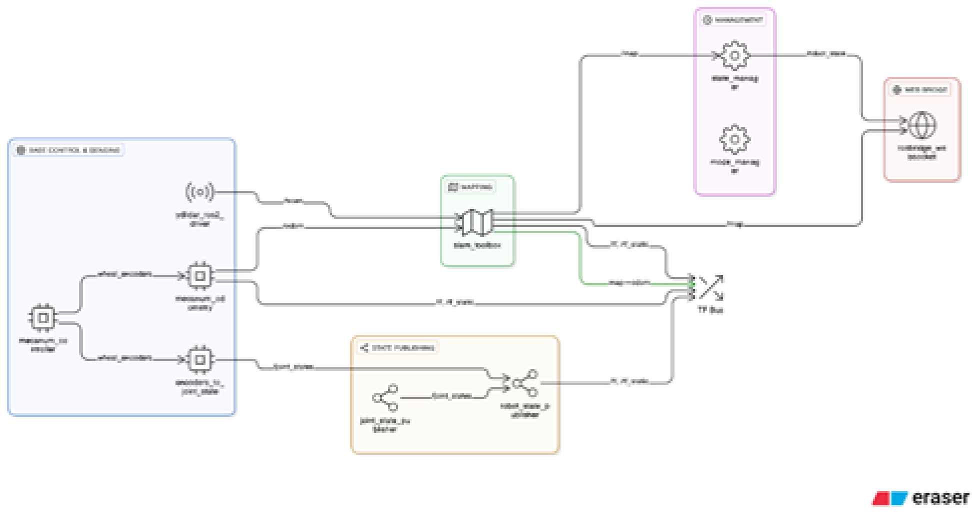


Fig 5 Node graph for navigation mode, with nav2 components (AMCL, planners, costmaps).

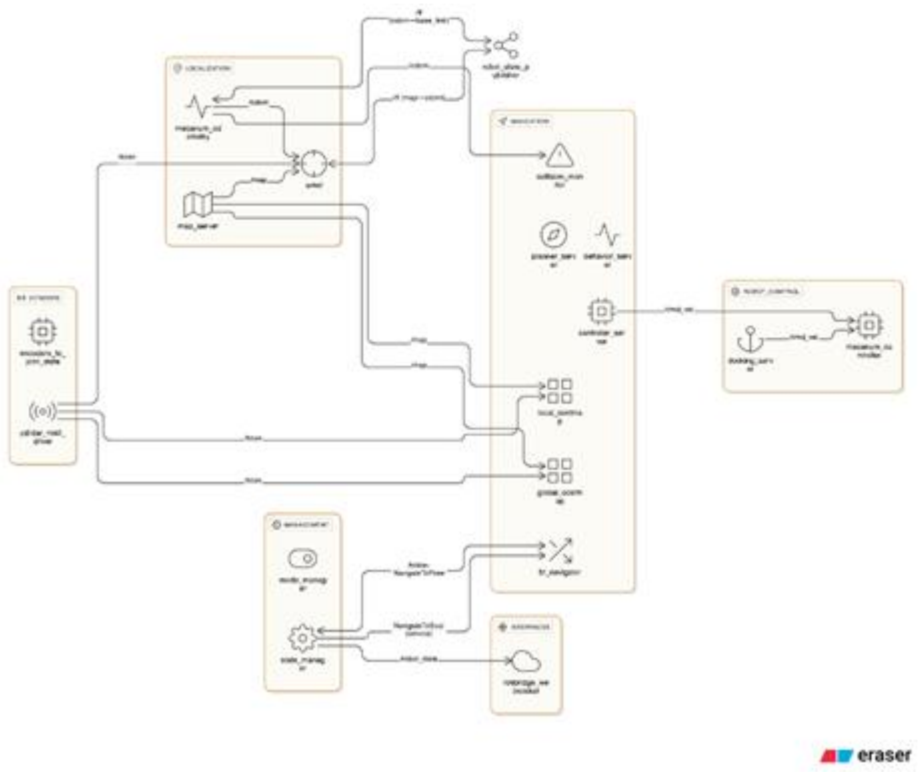


Fig. 6 ROS2 node graph (major nodes only) for mapping mode.

4.3 Concept Generation and Selection (I ignored the template layout because we have multiple solutions so we have multiple comparisons and instead I made it shorter…)

Early designs explored several alternatives for localization, docking, and user interaction. For global localization and navigation inside a mall parking facility, simple techniques such as line-following, manual waypoints, and RF or ultrasonic beacon-based navigation were considered but rejected:

- RF RSSI approaches suffered from severe multipath effects and delivered only decimeter-level accuracy (at best) in simulations and early tests, which is insufficient for WPT alignment.
- Ultrasonic localization would require permanent fixed beacons in the facility and careful synchronization, making deployment expensive and brittle in a real commercial garage.

These results motivated the choice of a LiDAR-centric solution:

- LiDAR SLAM (slam_toolbox) is used to map the environment once, producing a static occupancy grid with good geometric accuracy.
- Nav2 then uses this map, AMCL and costmaps for localization and path planning between arbitrary poses, making it easy to address changes in layout or add new parking spots without physical infrastructure.

The docking problem is treated separately. The system needs a stable way to "snap" to the exact charger position on a specific EV. For this, a camera-based approach using April-Tags was selected:

- April-Tags (family tag36h11) printed at a known physical size (e.g., 80 mm) are attached near each charging point. A dedicated script (tag_generator.py) generates tags with exact real-world dimensions given a DPI.
- The camera is calibrated using a chessboard and the camera_calibration.py script, which computes the intrinsic parameters (camera_matrix) and distortion coefficients (dist_coeffs) and saves them to camera_params.npz.
- The docking module detects the April-Tag, estimates its 3D pose relative to the camera, and derives linear and angular velocity commands to refine the robot's pose.[25]

Compared to alternatives (RF beacons, magnetic markers, or fixed mechanical guides), this approach combines low infrastructure cost (paper tags), high precision (pose estimation from vision), and flexibility (tags can be repositioned without rewiring).

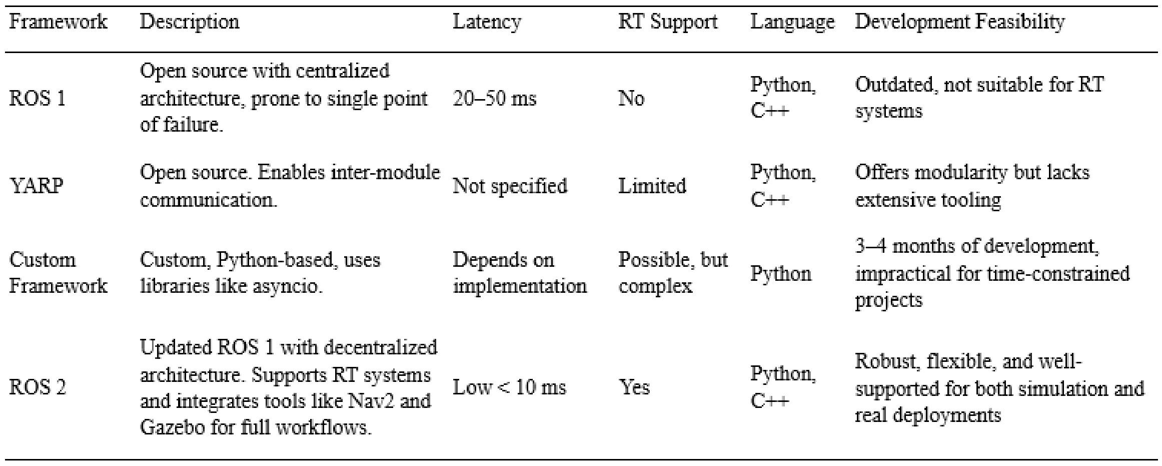
The user interface concept also underwent a transition. Instead of relying on RViz on an onboard screen, the final system uses Foxglove Studio for engineering-level visualization and a dedicated web dashboard for casual user operation. The dashboard communicates with ROS 2 through rosbridge_server and uses roslibjs in the browser. This design allows operators to access the robot from any laptop or tablet connected to the same network.

4.3.1 Morphological Matrix, Evaluation of Alternative Concepts / Trade Studies, Justification of Chosen Concept

Framework Selection

An appropriate software framework must be selected to enable communications and to act as a middleware between all other components. The framework serves as the middleware layer between embedded devices and the Raspberry Pi. Several frameworks were evaluated, ROS1, YARP, a custom Python-based system, and ROS2, each differing in latency, real-time support, modularity, and ease of development. Table below presents a comparison of these frameworks in terms of architecture, latency, real-time capabilities, supported languages, and overall development feasibility.

Table 9
Framework comparison



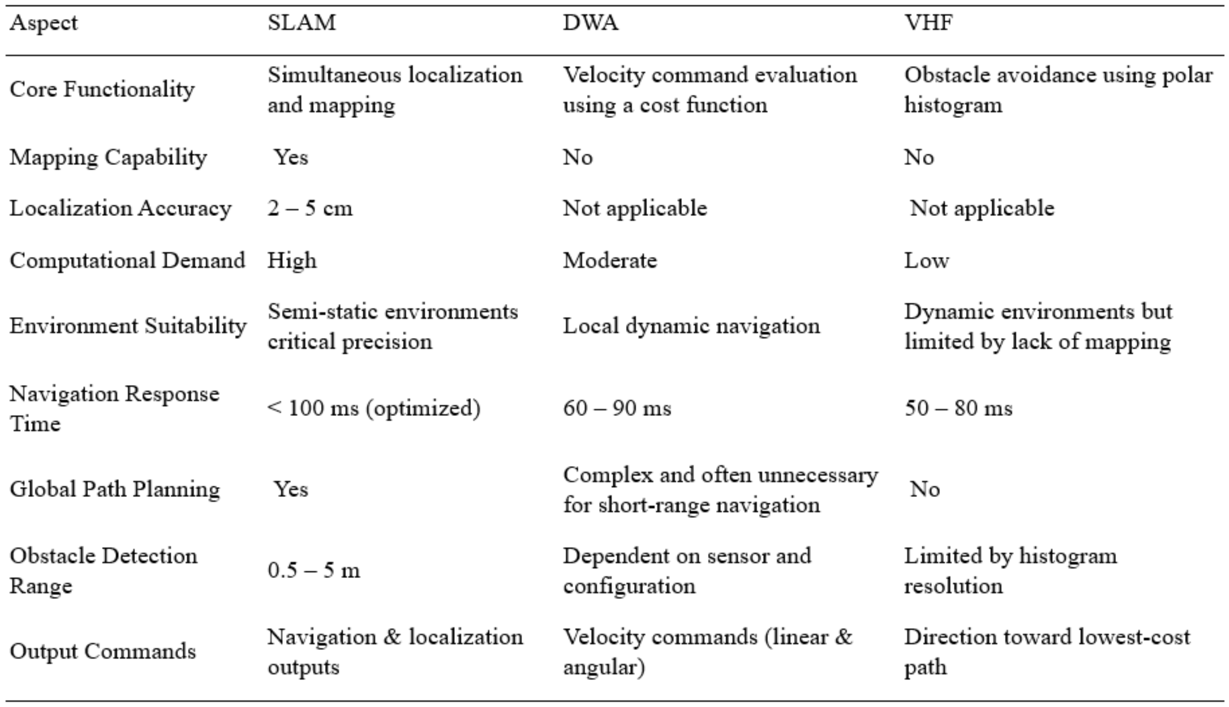
Framework	Description	Latency	RT Support	Language	Development Feasibility
ROS 1	Open source with centralized architecture, prone to single point of failure.	20–50 ms	No	Python, C++	Outdated, not suitable for RT systems
YARP	Open source. Enables inter-module communication.	Not specified	Limited	Python, C++	Offers modularity but lacks extensive tooling
Custom Framework	Custom, Python-based, uses libraries like asyncio.	Depends on implementation	Possible, but complex	Python	3–4 months of development, impractical for time-constrained projects
ROS 2	Updated ROS 1 with decentralized architecture. Supports RT systems and integrates tools like Nav2 and Gazebo for full workflows.	Low < 10 ms	Yes	Python, C++	Robust, flexible, and well-supported for both simulation and real deployments

Navigation Algorithm Selection

For the robot's motion planning and localization subsystem, our selected navigation algorithm must handle path generation, obstacle avoidance, and RT responsiveness, as it directly impacts how efficient and safe the robot can move within an environment while simultaneously receiving live localization data from the ESP32-based system. SLAM, DWA, and VFH were analyzed based on their mapping capability, localization accuracy, computational demand, and response time.

Table below presents a comparative summary of these algorithms, highlighting their suitability for different operating conditions.

Table 10
Navigation algorithm comparison



Aspect	SLAM	DWA	VHF
Core Functionality	Simultaneous localization and mapping	Velocity command evaluation using a cost function	Obstacle avoidance using polar histogram
Mapping Capability	Yes	No	No
Localization Accuracy	2 – 5 cm	Not applicable	Not applicable
Computational Demand	High	Moderate	Low
Environment Suitability	Semi-static environments critical precision	Local dynamic navigation	Dynamic environments but limited by lack of mapping
Navigation Response Time	< 100 ms (optimized)	60 – 90 ms	50 – 80 ms
Global Path Planning	Yes	Complex and often unnecessary for short-range navigation	No
Obstacle Detection Range	0.5 – 5 m	Dependent on sensor and configuration	Limited by histogram resolution
Output Commands	Navigation & localization outputs	Velocity commands (linear & angular)	Direction toward lowest-cost path

After evaluation, SLAM was selected as the primary navigation framework; as it provides both RT mapping and position estimation, which aligns perfectly with the RSSI localization system implemented.

Unlike DWA or VFH, which focus mainly on reactive motion control and obstacle avoidance, SLAM offers continuous map generation and high localization accuracy around 2–5 cm, crucial for demonstrating autonomous navigation performance on the Raspberry Pi with ROS2.

<u>Localization Technology Selection</u>

SLAM is used when the robot is exploring a new environment; it builds an occupancy-grid map while estimating the robot's pose in real time using LiDAR scans and wheel odometry, allowing the robot to create a complete representation of the space and localize itself within that map at the same time.

AMCL is used when a map has been uploaded, opposing SLAM, it does not rebuild the map; but instead it focuses only on localizing the robots inside the uploaded map; by using a particle filter to compare incoming LiDAR scans with the saved map, continuously correcting odometry drift and refining the robot's pose.

4.4 Modeling and Calculations

4.4.1 Mecanum Kinematics

The robot uses four mecanum wheels in an X configuration (front-left, front-right, rear-right, rear-left). The URDF (my_robot.urdf.xacro) defines base_link and four wheel links (wheel_fl, wheel_fr, wheel_rl, wheel_rr) attached by continuous joints at positions derived from the wheelbase and track:

- Wheelbase $= 0.26$ m (front–rear distance),
- Track $= 0.25$ m (left–right distance),
- Wheel radius $r = 0.0485$ m.

The MecanumController node implements the inverse kinematics. If the desired body-frame velocities are v_x (forward), v_y (left), and ω_z (yaw), the wheel linear velocities $v_{\text{fl}}, v_{\text{fr}}, v_{\text{rl}}, v_{\text{rr}}$ become:

$$\begin{aligned} v_{\text{fl}} &= v_x - v_y - (L_x + L_y)\,\omega_z \\ v_{\text{fr}} &= v_x + v_y + (L_x + L_y)\,\omega_z \\ v_{\text{rl}} &= v_x + v_y - (L_x + L_y)\,\omega_z \\ v_{\text{rr}} &= v_x - v_y + (L_x + L_y)\,\omega_z \end{aligned}$$

where L_x and L_y are half wheelbase and half track respectively. These continuous velocities are then converted to discrete commands for the closed-loop motor board, which expects signed bytes representing "pulses per 10 ms" per wheel. The controller scales velocities to these units using per-wheel scale factors (to compensate for manufacturing differences) and a global velocity_scale:

$$u_i = \text{clip}(\text{round}(v_i \cdot \text{velocity\\_scale} \cdot s_i), -100, 100)$$

with s_i the calibration scale for wheel i. The MecanumController also enforces acceleration limits to avoid sudden jerks that could degrade odometry.

4.4.2 Encoder-Based Odometry and TF

The MotorDriver speaks to a 4-channel DC motor/encoder board over I²C, reading cumulative encoder counts via register REG_ENCODER_TOTAL (0x3C) and writing speed commands to REG_MOTOR_FIXED_SPEED (0x33). The opaque board returns per-wheel signed 32-bit tick totals, which are published by mecanum_controller as Int32MultiArray on wheel_encoders.

The TPRMeasurement tool (measure_tpr) is used to experimentally estimate the number of ticks per mechanical revolution (TPR). The operator observes initial encoder values, rotates one wheel exactly one revolution, and computes the differences. This measurement is repeated several times to obtain a reliable TPR (e.g., 2400 ticks/rev, including gearbox).

The MecanumOdometry node subscribes to wheel_encoders, computes increments Δn_i per wheel, converts them into angles $\Delta\phi_i$ using:

$$\Delta\phi_i = 2\pi \frac{\Delta n_i}{\mathrm{TPR}}$$

and then into body-frame displacements $\Delta x_b, \Delta y_b, \Delta\theta$ using the forward kinematics:

$$\begin{aligned}
\Delta x_b &= \frac{r}{4}(\Delta\phi_{\mathrm{fl}} + \Delta\phi_{\mathrm{fr}} + \Delta\phi_{\mathrm{rl}} + \Delta\phi_{\mathrm{rr}}) \\
\Delta y_b &= -\frac{r}{4}(-\Delta\phi_{\mathrm{fl}} + \Delta\phi_{\mathrm{fr}} + \Delta\phi_{\mathrm{rl}} - \Delta\phi_{\mathrm{rr}}) \\
\Delta\theta &= \frac{r}{4(L_x + L_y)}(-\Delta\phi_{\mathrm{fl}} + \Delta\phi_{\mathrm{fr}} - \Delta\phi_{\mathrm{rl}} + \Delta\phi_{\mathrm{rr}})
\end{aligned}$$

These body-frame increments are transformed into the map/odom frame using the current heading θ. The odometry pose (x, y, θ) is then published as nav_msgs/Odometry on /odom, and the node simultaneously broadcasts a TF transform from odom to base_link.

The TF tree thus has the structure:

$$\text{map} \xrightarrow{\text{slam\_toolbox \& nav2}} \text{odom} \xrightarrow{\text{mecanum\_odometry}} \text{base\_link} \xrightarrow{\text{robot\_state\_publisher}} \text{(wheels \& LiDAR)}$$

slam_toolbox (in mapping mode) and AMCL (in navigation mode) compute and broadcast the map → odom transform. The URDF's laser_joint defines a fixed transform between base_link and laser_frame corresponding to the physical LiDAR mount. A view_frames snapshot confirms this TF tree and transform frequencies.

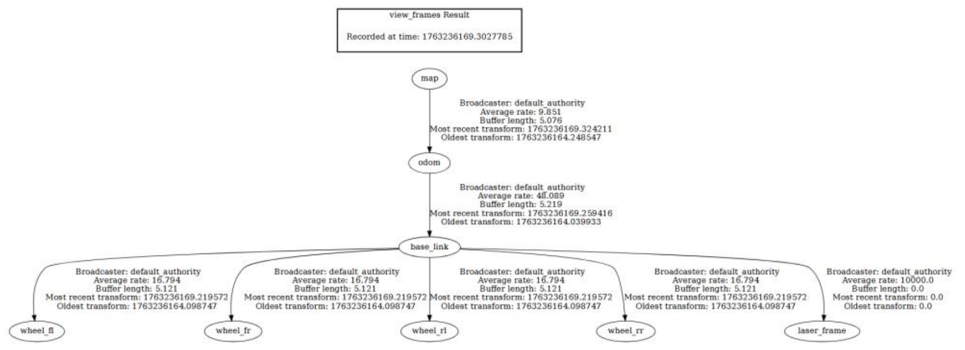

view_frames Result
Recorded at time: 1763236169.3027785
map
Broadcaster: default_authority
Average rate: 9.851
Buffer length: 5.076
Most recent transform: 1763236169.324211
Oldest transform: 1763236164.248547
odom
Broadcaster: default_authority
Average rate: 48.089
Buffer length: 5.219
Most recent transform: 1763236169.259416
Oldest transform: 1763236164.039933
base_link
Broadcaster: default_authority
Average rate: 16.794
Buffer length: 5.121
Most recent transform: 1763236169.219572
Oldest transform: 1763236164.098747
Broadcaster: default_authority
Average rate: 16.794
Buffer length: 5.121
Most recent transform: 1763236169.219572
Oldest transform: 1763236164.098747
Broadcaster: default_authority
Average rate: 16.794
Buffer length: 5.121
Most recent transform: 1763236169.219572
Oldest transform: 1763236164.098747
Broadcaster: default_authority
Average rate: 16.794
Buffer length: 5.121
Most recent transform: 1763236169.219572
Oldest transform: 1763236164.098747
Broadcaster: default_authority
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
wheel_fl
wheel_fr
wheel_rl
wheel_rr
laser_frame

Fig. 7 TF tree visualization (map → odom → base_link → wheels, laser_frame) generated from view_frames.

4.4.3 Camera Pose and Docking Control

The camera calibration script calculates intrinsic parameters K and distortion coefficients d. For docking, the AprilTag detector (cv2.aruco with family tag36h11) is used. For a tag of known size s (e.g., 0.08 m), estimatePoseSingleMarkers returns:

- Rotation vector $\mathbf{r}$,
- Translation vector $\mathbf{t} = (x, y, z)^\top$ in the camera frame.

The prototype docking script converts these into simple proportional control commands, where z is used as a distance estimate and x as a lateral/yaw misalignment proxy:

$$v = -k_v(z - z_{\text{target}})$$
$$\omega = -k_\omega x$$

with k_v and k_ω tuned gains and z_{target} the desired standoff distance between camera and tag. In the final system, this logic will be implemented as a ROS 2 node (camera_docking_node) that publishes geometry_msgs/Twist to /cmd_vel, layering on top of Nav2 for the final centimetre-level alignment.

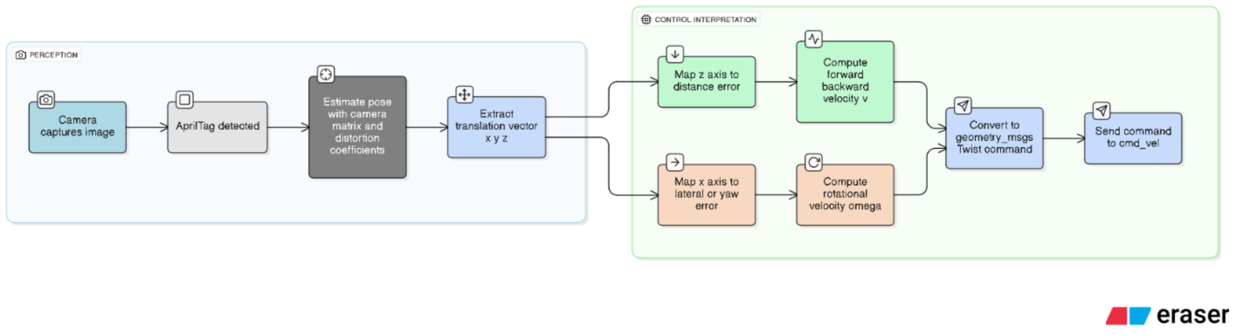

PERCEPTION
Camera captures image
AprilTag detected
Estimate pose with camera matrix and distortion coefficients
Extract translation vector x y z
CONTROL INTERPRETATION
Map z axis to distance error
Compute forward backward velocity v
Map x axis to lateral or yaw error
Compute rotational velocity omega
Convert to geometry_msgs Twist command
Send command to cmd_vel
eraser

Fig. 8 April-Tag detection process.

4.5 Hardware Design Approach (Circuit Design, Components, PCB Layout)

The hardware design had to satisfy three constraints: carrying capacity for a future full-scale WPT pad, sufficient sensor fidelity (LiDAR and camera), and robustness appropriate for real parking floors.

The base is a steel chassis rated for up to approximately 25 kg, with mounting points for the four motors and mecanum wheels, the LiDAR, the camera, and the WPT pad. The motors are connected to a dedicated driver board with integrated encoders, which simplifies wiring and ensures synchronized reading of all four encoders.

The YDLIDAR G4 is mounted at the front of the robot at a height that avoids occlusion by the chassis and wheels, with a clear 360° line of sight. The LiDAR is powered through a regulated supply and driven via ydlidar_ros2_driver, which itself wraps the vendor's ydlidar_sdk.

The RGB camera is placed on a small mast close to the front, tilted to capture the region where the WPT pad will be when the robot is close to the EV. The mounting and height are chosen such that the AprilTag can be detected reliably at distances of roughly 0.3–1.0 m.

The commercial WPT transmitter pad is mounted centrally in the front of the chassis. Mechanically, the intention is for the pad center to coincide with the origin of the robot's base_link frame in the URDF, so that the AprilTag pose and docking controller can be tuned to achieve the desired coil alignment tolerance (e.g., ±2–3 cm sideways and front-back).

Power distribution is handled by a DC bus feeding the motors and a separate regulated bus for logic (Pi 5, LiDAR, camera, ESP32 modules). The EV-side ESP32 is powered from the charger's low-voltage rail and monitors voltage and current via appropriate sensors before sending telemetry via ESP-NOW.

4.6 Software Design Approach (if applicable) (programming languages, algorithms, flowcharts).

4.6.1 High-Level Functional Model

The System can be viewed as a pipeline from **environment and user inputs** to **actuator commands and user feedback**.

At a high level:

- The **environment** (parking lanes, parked EVs, obstacles, QR/AprilTags, charger status) is sensed by:
    - LiDAR (2D point cloud as /scan)
    - Wheel encoders (wheel_encoders)
    - The camera (image stream)
    - EV-side current and voltage sensors (via ESP-NOW)
- The **user** interacts through:
    - The web dashboard (mapping, parking definition, mission commands)
    - A PS5 controller or on-screen joystick (for manual mapping and fine positioning)

These inputs are processed in several stages:

1. **Base Control & Odometry Layer**
   The mecanum_controller and mecanum_odometry nodes convert desired velocities into motor commands and reconstruct the robot's motion from encoder ticks. They also maintain the TF transform odom → base_link.

2. **Perception & Localization Layer**
   The LiDAR driver (ydlidar_ros2_driver) publishes scans. During mapping, slam_toolbox consumes /scan and /odom to build a map and publish map → odom. During navigation, AMCL consumes /scan, /map, and /odom to localize the robot and maintain the same map → odom transform. The URDF and robot_state_publisher provide static transforms from base_link to wheels and laser_frame.

3. **Planning & Navigation Layer (Nav2)**
   In navigation mode, the Nav2 stack (planner_server, controller_server, bt_navigator, costmaps) uses the map and localization to compute and execute paths to goal poses. It outputs cmd_vel commands, which feed back into the base control layer.

4. **Docking & Alignment Layer (Camera)**
   When the robot is close to a selected parking spot, a camera node detects the AprilTag near the charger, estimates its pose, and computes small velocity corrections to achieve precise alignment. This stage refines the Nav2 goal to meet wireless charging alignment tolerances.

5. **Mission & Map Management Layer**
   The state_manager and mode_manager nodes orchestrate system modes (mapping vs navigation), map saving/loading, and parking spot management. They provide high-level services used by the dashboard and publish a consolidated /robot_state topic.

6. **User Interface & Monitoring Layer**
   The Node.js web server and rosbridge_server allow the dashboard to:

   - Display maps, robot pose, parking spots, and system state.
   - Issue mode changes, navigation missions, and parking spot saves.
   - Show charging metrics and alerts.

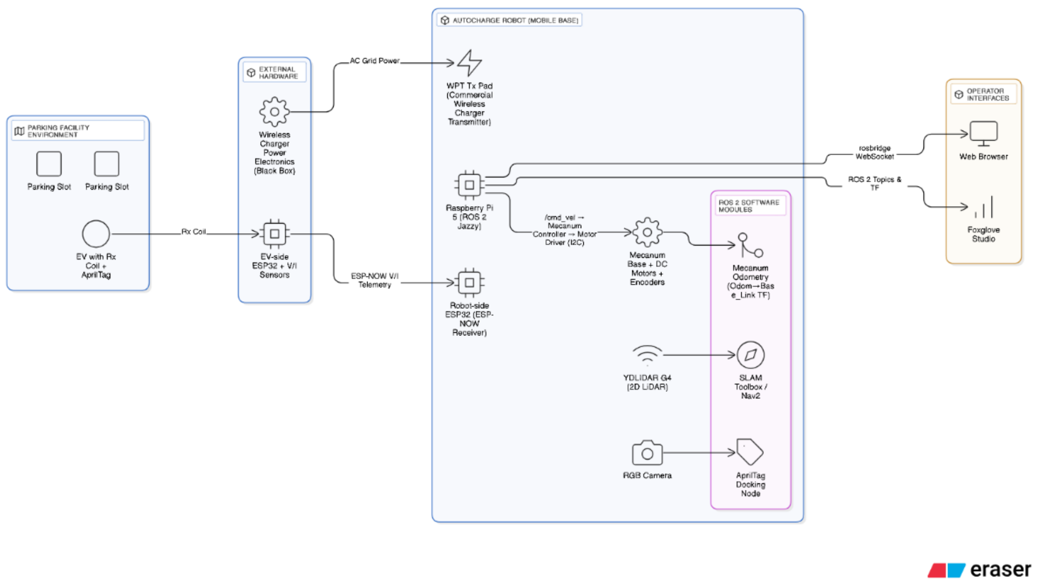

PARKING FACILITY ENVIRONMENT
Parking Slot
Parking Slot
EV with Rx Coil + AprilTag
Rx Coil
EXTERNAL HARDWARE
Wireless Charger Power Electronics (Black Box)
EV-side ESP32 + V/I Sensors
AC Grid Power
ESP-NOW V/I Telemetry
AUTOCHARGE ROBOT (MOBILE BASE)
WPT Tx Pad (Commercial Wireless Charger Transmitter)
Raspberry Pi 5 (ROS 2 Jazzy)
Robot-side ESP32 (ESP-NOW Receiver)
/cmd_vel → Mecanum Controller → Motor Driver (I2C)
Mecanum Base + DC Motors + Encoders
YDLIDAR G4 (2D LiDAR)
RGB Camera
ROS 2 SOFTWARE MODULES
Mecanum Odometry (Odom→Base_Link TF)
SLAM Toolbox / Nav2
AprilTag Docking Node
rosbridge WebSocket
ROS 2 Topics & TF
OPERATOR INTERFACES
Web Browser
Foxglove Studio
eraser

Fig. 9 Full system block diagram

4.6.2 Inputs, Internal Processing, and Outputs

Functionally, the Software behaves like a pipeline that starts with sensing the environment and user intentions, transforms these into decisions and motion commands, and then feeds rich state and charging information back to the operator.

At the input side, the robot perceives its surroundings through the LiDAR, wheel encoders, and camera, while the charging subsystem exposes voltage and current through the EV-side ESP32 over ESP-NOW. The LiDAR publishes a continuous stream of 2D scans on `/scan`, capturing walls, cars, and obstacles. The motor controller reads integrated encoder counts from the I²C motor board and republishes them as `wheel_encoders`. In parallel, the camera delivers RGB frames that will later be used by the docking node to detect the AprilTag marker near the charger. On the user side, the dashboard sends high-level commands such as "switch to mapping mode", "save this map as floor_A", "remember the current pose as parking spot B1-3", or "navigate to spot S7", all implemented as service calls to the `state_manager`. For mapping and fine positioning, the PS5 controller and the on-screen joystick generate `cmd_vel` messages that go directly to the drive controller.

Inside the system, these inputs are absorbed by several cooperating processing blocks. At the bottom, `mecanum_controller` and `mecanum_odometry` form the base control layer. They are the interface between abstract velocities and physical movement: `mecanum_controller` receives `cmd_vel` from teleop, Nav2 or the docking controller, applies the mecanum inverse kinematics and per-wheel calibration, and sends appropriate speed commands over I²C to the motor board. The resulting wheel motions are reflected in updated encoder counts, which `mecanum_odometry` turns into pose increments and publishes as `/odom` along with a TF transform from `odom` to `base_link`. The same encoder data is converted into joint angles by `encoders_to_joint_state`, which allows the wheels in the URDF model to rotate correctly in Foxglove.

Above this base layer sits the perception and localization logic. In mapping mode, `slam_toolbox` consumes `/scan` from the YDLIDAR driver and `/odom` from the odometry node to construct an occupancy grid map of the environment and to maintain a `map → odom` transform. In navigation mode, a static map server loads a previously saved YAML map, and AMCL replaces SLAM as the source of `map → odom`, performing particle filter localization using `/scan`, `/map` and `/odom`. The URDF and `robot_state_publisher` provide the static transforms from `base_link` to the LiDAR (`laser_frame`) and the four wheels, so all sensing and motion are expressed consistently in the TF tree.

Once the robot has a pose in the map frame, the Nav2 stack is able to plan and execute motions. The planner and controller nodes, configured via `nav2_params.yaml`, read `/map`, the robot's current pose (via TF), and LiDAR-based costmaps, and then compute feasible paths to goal poses. The behavior tree navigator orchestrates global planning, local path following, and recovery behaviors (for example, clearing costmaps and spinning when stuck). Its primary output is a stream of `cmd_vel` commands that direct the mecanum base along the planned path while respecting obstacles. Close to the end of the trajectory, a dedicated docking controller built around the camera takes over. It reads camera images, detects the AprilTag attached near the charger, estimates its pose using the calibration parameters, and generates small velocity corrections to eliminate remaining distance and lateral/yaw errors. This refined `cmd_vel` stream ensures that the robot's WPT pad is brought into the narrow alignment window required by the wireless charger.

In parallel, the system continuously manages maps, modes, and missions. The `state_manager` node receives dashboard requests to change mode, save or list maps, store or list parking spots, and dispatch navigation goals. It forwards mode changes to `mode_manager`, which starts or stops SLAM Toolbox and Nav2 as separate processes in the correct order. When a map is saved, `state_manager` invokes the Nav2 map saver and writes the occupancy grid and YAML description into the maps directory. When a parking spot is stored, it captures the current pose in the map frame from TF and appends a labeled entry to a JSON file associated with that map. At all times it publishes a unified `/robot_state` message containing the current mode, active map, whether Nav2 is ready, and the robot's pose. This single topic drives much of the dashboard's state display.

The charging telemetry path is conceptually similar. The EV-side ESP32 samples voltage and current from the charger output, encodes them into small ESP-NOW packets, and broadcasts them wirelessly. The robot-side ESP32 receives these frames and passes them over a serial link to a bridge node on the Pi. That node parses the data and publishes ROS topics such as `/wpt/voltage`, `/wpt/current` and `/wpt/power`. These topics are consumed purely by monitoring components: the dashboard uses them to display real-time charging metrics, and in future the docking or mission logic could use them to determine when charging is complete or to detect abnormal conditions.

On the output side, this processing pipeline manifests in three main ways. First, it generates **physical motion**: the combined effect of teleop, Nav2, and docking controllers on `cmd_vel` leads to motor commands on the I²C bus and corresponding robot trajectories in the parking facility. Second, it produces **persistent data artifacts** in the form of map files and parking spot definitions, which capture the structure and semantics of each environment and can be recalled later for autonomous operation. Third, it provides **rich feedback** to humans and monitoring tools: the dashboard renders the live map, robot pose, saved parking spots, power metrics and alerts by subscribing to `/map`, `/robot_state`, `/odom`, `/wpt/*` and battery topics via `rosbridge_server`, while Foxglove Studio visualizes the same data directly from ROS. In this way, every major input—from LiDAR scans and encoder ticks to user commands and charger measurements—flows through a chain of deterministic transformations and emerges as either safe robot motion, durable configuration (maps and parking metadata), or intelligible status information for the operator.

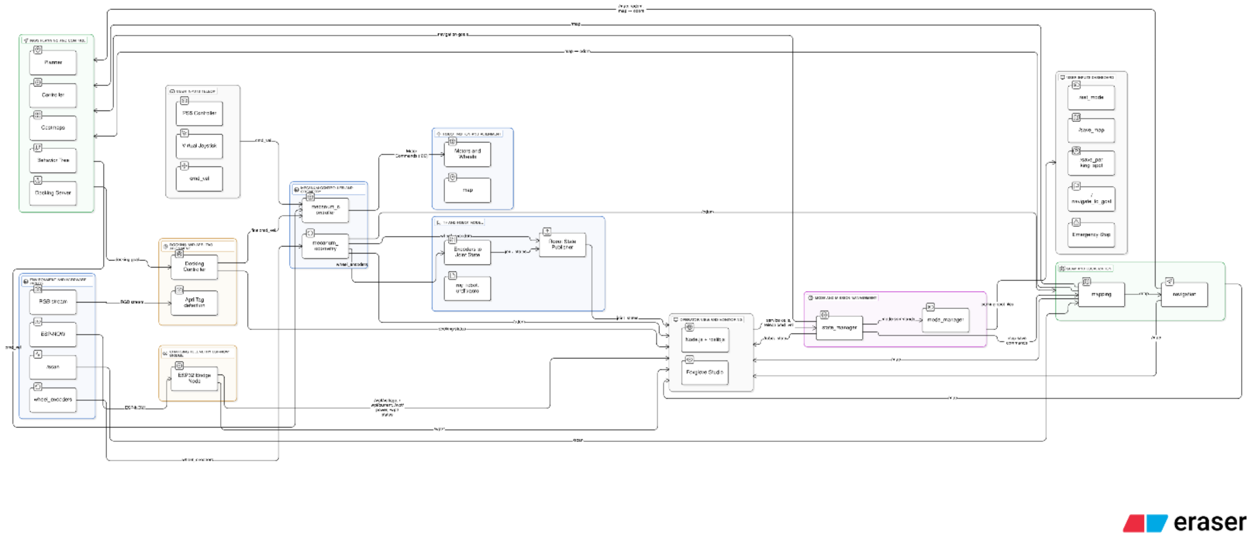


Fig. 10 Flow of data between nodes from inputs to outputs.

4.6.3 Control Flow in Typical Scenarios

To clarify control flow, consider two representative scenarios.

Scenario A – Mapping a New Parking Area

1. Operator opens the Setup page and switches to **mapping** mode.
    - The dashboard calls /set_mode with mode="mapping".
    - state_manager forwards this to mode_manager, which starts slam_toolbox and stops Nav2.
2. Operator uses the joystick or PS5 controller to drive the robot slowly around the area.
    - Dashboard or teleop publishes cmd_vel.
    - mecanum_controller sends motor commands; mecanum_odometry updates /odom.
3. slam_toolbox receives /scan and /odom and builds a map in real time, publishing /map and map → odom.
    - Dashboard shows the map and robot pose by subscribing to /map and /robot_state.
4. Once satisfied, operator enters a map name and clicks "Save Map".
    - Dashboard calls /save_map.
    - state_manager calls map_saver_cli to write <name>.yaml to disk.

Scenario B – Navigating to a Parking Spot and Docking

1. Operator switches to **navigation** mode and selects a map.
    - Dashboard calls /set_mode with mode="navigation", map_name="<name>".
    - mode_manager stops SLAM, launches AMCL + Nav2 with that map.
2. Operator selects a parking spot from the list.
    - Dashboard calls /list_parking_spots to populate the list, then /navigate_to_goal with the chosen spot's pose.
3. state_manager sends a NavigateToPose goal to Nav2.
    - Nav2 planners and controllers compute a path and stream cmd_vel to mecanum_controller.

4. When Nav2 reaches the goal region (inside xy and yaw tolerances), the docking block is activated.

   - The camera detects the AprilTag; the docking node computes fine velocity corrections until the alignment thresholds are met.

5. Once aligned, the WPT charger is enabled.

   - EV-side ESP32 sends voltage/current data via ESP-NOW; the robot-side bridge publishes WPT metrics.

   - Dashboard displays charging status and allows the operator to stop charging or send the robot back to a base station.

In both scenarios, the **control flow** proceeds from high-level commands (via services or actions) down to cmd_vel, then to the motor board, while **feedback** flows from sensors back up to SLAM/AMCL/Nav2 and finally to the dashboard and Foxglove.

4.7 List of Required Components (Hardware & Software)

Table 11
List of required components

Components	Reason of use
Metal chassis frame	Strong support to all mounted components ensuring stable motion and accurate sensor readings
Mecanum wheel robot kit (with 12V DC encoder motors)	For omnidirectional mobility and stability for smooth and precise movement
ESP32	For communication between Tx on robot and Rx on test car
Wireless charger module	Enable contactless energy transfer to stimulate autonomous EV charging
BMS	Protects from overcurrent, overvoltage, and discharge during wireless charging test for safety
Voltage/ current sensor	Monitor charging performance of the WPTS; sends readings to raspberry pi for display and dashboard visualization
Li-ion battery	Powers the robot and serves as the energy storage system during the wireless charging tests simulating a scaled down battery
Webcam	Next step of alignment to further increase accuracy of alignment

4.8 Project Planning (Work Packages, Timeline & Milestones)

Table 12
WBS

Work package	Tasks	Deliverables
WP1 – Project planning and research (Capstone 1)	1. Conduct a literature review on wireless EV charging, autonomous docking, SLAM, Nav2, mecanum bases, and ESP-NOW. 2. Define project objectives, scope, success criteria, assumptions, and constraints. 3. Prepare initial schedule, budget estimate, and risk assessment with mitigation actions.	1. Summary report on related work and enabling technologies. 2. Documented problem statement, objectives, success metrics, and project scope. 3. Approved project plan including schedule, budget, and risk register.
WP2 – Simulation and software planning (Capstone 1)	1. Define the ROS 2 node architecture, topics, services, and TF frames for the system. 2. Build a simulation model of the robot in Gazebo, including mecanum base and LiDAR. 3. Prototype SLAM + Nav2 navigation in a virtual parking map and simulate basic docking scenarios.	1. Architecture diagram and interface specification for ROS 2 nodes and TF tree. 2. Gazebo simulation model of the robot and sample parking environment. 3. Simulation results demonstrating basic mapping, navigation, and parking workflows.
WP3 – Robot base assembly and odometry (Capstone 1)	1. Assemble the mecanum chassis and mount DC motors, encoders, and electronics plate. 2. Implement and test the I²C motor driver and mecanum_controller node, including PS5 teleoperation. 3. Implement mecanum_odometry, measure ticks-per-revolution (TPR), and validate /odom and TF.	1. Fully assembled base robot with mecanum drive and mounted electronics. 2. Working low-level motor control with encoder feedback and teleop using ROS 2. 3. Validated odometry and TF chain odom → base_link, suitable for SLAM and Nav2.
WP4 – LiDAR integration and SLAM mapping (Capstone 1)	1. Mount and wire the YDLIDAR sensor; verify /scan and laser_frame alignment. 2. Configure SLAM Toolbox to build accurate 2D occupancy maps using /scan and /odom. 3. Implement map saving and basic multi-map handling for future navigation.	1. Integrated LiDAR sensor with correct TF transform to base_link. 2. Initial occupancy maps of the lab/parking-like environment with loop closures. 3. Saved map files (YAML + image) and procedure for creating multiple maps.

WP5 – Capstone 1 documentation and review	1. Compile Capstone 1 report covering background, methodology, simulations, and preliminary hardware results. 2. Prepare and deliver Capstone 1 presentation; capture feedback for Capstone 2.	1. Capstone 1 written report. 2. Presentation slides and recorded feedback notes to refine Capstone 2 work.
WP6 – Navigation stack and mission management (Capstone 2)	1. Configure AMCL and Nav2 on the physical robot using maps produced in Capstone 1. 2. Implement state_manager and mode_manager for switching between mapping and navigation and managing maps and parking spots.	1. Robot localized and navigating autonomously to goals within the saved maps. 2. Mode management framework with services for /set_mode, /save_map, /save_parking_spot, /list_parking_spots, and /navigate_to_goal.
WP7 – Dashboard, docking and ESP-NOW integration (Capstone 2)	1. Develop the AutoCharge web dashboard (Dashboard & Setup pages) with ROSBridge integration for teleop, map view, and mission control. 2. Prototype AprilTag-based docking: calibrate camera, detect tag, and generate fine cmd_vel corrections near the parking spot. 3. Integrate ESP-NOW telemetry: connect EV-side ESP32 to voltage/current sensors and bridge data to ROS topics and the dashboard.	1. Web-based operator UI for mapping, navigation, parking definition, and emergency stop. 2. Docking prototype demonstrating centimeter-level alignment using camera and tag pose. 3. Live charging metrics (voltage, current, power) available as ROS topics and visible on the dashboard.
WP8 – System testing, validation and final reporting (Capstone 2)	1. Perform end-to-end tests: create maps, switch between maps, navigate to parking spots, and execute docking while monitoring telemetry. 2. Evaluate performance (goal accuracy, docking precision, robustness) under different layouts and surfaces. 3. Prepare final Capstone 2 documentation, including test results, analysis, and future work.	1. Test and validation report summarizing mapping, navigation, docking, and telemetry performance. 2. Quantitative and qualitative evaluation of the prototype against objectives. 3. Final Capstone 2 report, presentation slides, and demo plan.

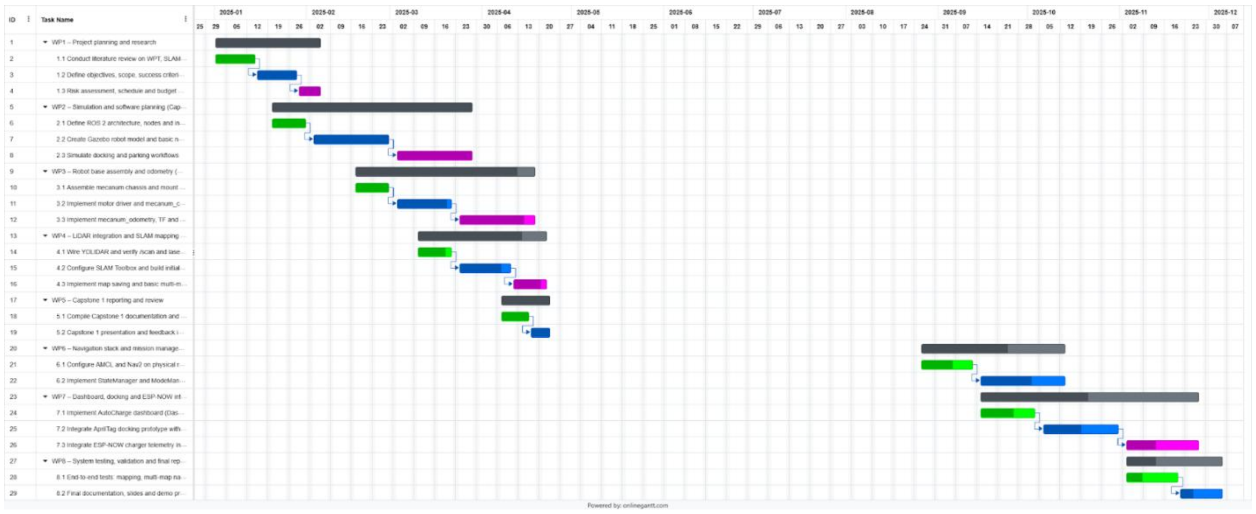


Fig. 11 Gantt Chart

4.9 Project Budget

Table 13
Components and budget

Item	Quantity	Unit Cost (QAR)	Total Cost (QAR)
Raspberry Pi 5	1	473.00	473.00
YDLIDAR G4	1	713.60	713.60
Mecanum wheel robot car kit with 12V DC hall encoder motor arduino	1	276.14	276.14
5V 2A Wireless Charging Tx/ Rx Modules	1	49.00	49.00
Li-ion battery 3.7V 3800mah - 18650	6	20.00	120.00
3.7V Li-ion Battery Charger	2	34.00	68.00
8650 Battery Charger Protection Board (BMS)	1	15.00	15.00
Resistors	2	5.00	10.00

Others (Wires, etc.)	-	-	50
Total			**1774.14 QAR**

4.10 Project Management (Team Structure, Roles, and Responsibilities)

Nadine Al-Jada – (Role: Hardware & integration lead)

- Assemble mecanum robot base, installed motors, encoders, LiDAR, camera, and wireless charging components.
- Integrated all sensors with RPi, ensured correct wiring, power distribution, and mechanical mounting.
- Hardware troubleshooting, encoder calibration, LiDAR alignment, and camera field-of-view optimization.
- Supported system testing, particularly docking tests and ESP-NOW telemetry validation.

Islam Azzam – (Role: Software & navigation lead)

- Designed and implemented the ROS2 architecture, including motor control nodes, odometry, SLAM Toolbox, AMCL, and Nav2 configuration.
- Developed the state_manager, mode_manager, and map/parking-spot management services.
- Implemented the AprilTag docking scripts and dashboard communication through rosbridge and Node.js.
- Led the software debugging process, map generation, navigation tuning, and integration of all subsystems.

Shared Responsibilities

- Defined project objectives, success criteria, and risk mitigation strategies.
- Conducted literature review, concept selection, and comparison studies.
- Perform end-to-end testing, mapping, navigation, docking, and charging telemetry monitoring.
- Co-created the web dashboard interface, prepared documentation, and delivered capstone presentations.

4.11 Expected Deliverables: (What the project will produce at the end of Capstone 1 and Capstone 2)

The project is expected to deliver a fully functional autonomous wireless charging robot capable of mapping its environment, localizing within uploaded maps, navigating to parking spots, and performing centimeter-level alignment with an EV's wireless charging pad. The final prototype includes a mobile base, a 2D YDLIDAR for SLAM-based mapping and AMCL localization, and a calibrated RGB camera for AprilTag-based fine docking. A RPi 5 running ROS2 Jazzy handles all major computation, including map generation, Nav2 path planning, obstacle avoidance, and docking corrections. On the EV side, a compact ESP32 module measures charging voltage and current and transmits real-time telemetry to the robot through ESP-NOW.

In addition, the project will deliver a complete software framework integrating ROS2 nodes for motor control, odometry, SLAM, navigation, map management, docking, and telemetry monitoring. A custom web dashboard provides manual joystick control, map visualization, parking-spot management, emergency stop, and live system status, including power flow during charging. Comprehensive testing results covering mapping accuracy, navigation consistency, docking precision, and telemetry responsiveness will be documented to validate overall system performance and demonstrate the feasibility of autonomous wireless EV charging

5 Chapter 5: Implementation

Executed in a multi-stage process to ensure mechanical integrity and safety, allowing for systematic verification at each stage before proceeding to the next

5.1 Hardware Implementation (Component assembly, soldering, 3D printing, PCB fabrication, etc.)

First, the mechanical base and drive system were built using four 12V DC motors with mecanum wheels arranged in X configuration, mounted at chassis corners, respecting both wheelbase $2L_x$ and track width $2L_\gamma$ dimensions defined in URDF; crucial for accurate kinematics calculations and odometry. The motor/encoder board was mounted on an internal plate, with short wiring runs to each motor to reduce noise, voltage drop and reduce EMI.

RPi 5, LiDAR, and camera were then installed. The LiDAR was attached with a simple bracket near the front of the robot, its center aligned with the `laser_frame` origin in the URDF. The camera was mounted slightly above the LiDAR, looking downwards, optimizes the field of view for detecting the QR code to start the final docking stage.

For stable operation, power wiring was organised into separate bundles for high-current motor lines and low-current sensor/logic lines. Simple EMI precautions were taken, such as twisting motor leads and routing them away from LiDAR and camera cables. The EV-side ESP32, voltage sensor, and current sensor were wired to the commercial charger output. The ESP-NOW link was verified on the bench with simulated loads.

For safety, an emergency stop button was wired inline with the motor supply. Pressing this cuts power to the motors, independent of software, while still allowing the Pi 5 and sensors to run and report the state over Wi-Fi.

5.2 Software Development (Finalized code, debugging, and optimizations)

Executed using a modular approach within the ROS 2 Jazzy framework; ensuring robust communication between nodes, reliable sensor data processing, and perfect control logic for autonomous operation.

5.2.1 Base Bringup and Teleoperation

5.2.2 TF Verification and Robot Description

5.2.3 SLAM Toolbox and Map Saving

5.2.4 Navigation Stack Bringup and Tuning

5.2.5 Parking Spot Management and Testing

5.2.6 AprilTag Docking Experiments

Parallel to the navigation work, we developed and tested AprilTag-based docking using standalone Python scripts before fully integrating it into ROS2. The following tools were prepared:

a. camera_calibration.py

Used to collect multiple chessboard images and compute the camera_matrix and dist_coeffs, which correct lens distortion and allow accurate pose estimation.

b. tag_generator.py

Generated AprilTag tag36h11 markers at controlled sizes (120 mm), ensured the printed tag size exactly matched the 'TAG_SIZE' parameter used in the docking logic.

c. april_tag_nav.py

A test script that used OpenCV's AprilTag/ArUco detection to detect the tag in the live video stream, estimate its 3D pose relative to the camera, and compute simple velocity corrections (forward and yaw) toward a target distance TARGET_DIST.

These experiments confirmed that a single AprilTag provides stable depth and lateral-error feedback, and that we could reliably compute how far the robot is from the tag and whether it is left/right of center.

The next step is to wrap this logic into a ROS2 docking node that:

- subscribes to a camera topic,
- publishes tag pose estimates as ROS messages,
- and publishes fine cmd_vel corrections once Nav2 reaches the coarse goal.

In the final architecture, this docking logic becomes part of the camera package, taking over during the final alignment phase that happens after Nav2 navigation.

5.2.7 Dashboard Integration and ROSBridge

5.2.8 Full system verification

5.3 Prototype Development (Final version with step-by-step assembly details)

This section briefly summarizes how the final prototype was built and brought to a fully working state.

*5.3.1 **Mechanical Assembly***

The build began with a metal mecanum chassis rated for about 25 kg. Four DC motors with encoders and mecanum wheels were mounted in an X-configuration, matching the URDF geometry (wheelbase ≈ 0.26 m, track ≈ 0.25 m, wheel radius ≈ 0.0485 m). Care was taken to ensure all wheels touched the ground evenly.

A bracket at the front center held the 2D LiDAR (YDLIDAR G4) at the height and offset specified in the URDF `laser_frame`. Inside the chassis, the Raspberry Pi 5, the motor/encoder board, power distribution, and the robot-side ESP32 were fixed on standoffs. A small mast at the front carried the RGB camera, angled so that an AprilTag near the charger would be visible at docking distance. Finally, the commercial WPT transmitter pad was bolted under the chassis so its center lined up with the `base_link` origin.

5.3.2 Electrical Wiring

Power distribution was split into a high-current rail (motors, driver) and a low-current rail (Pi 5, LiDAR, camera, ESP32s), with DC–DC converters and inline fuses. Each motor was wired to the motor/encoder board, which in turn connected to the Pi via I²C (SDA/SCL + common ground).

The LiDAR and camera were connected over USB (or CSI for the camera). On the EV/charger side, a second ESP32 was wired to voltage and current sensors on the charger output. On the robot, the ESP32 receiver forwarded ESP-NOW frames to the Pi over UART/USB. A hardware emergency-stop switch was placed inline with the motor supply so motors could be cut without shutting down the Pi and sensors.

5.3.3 Base and URDF Bring-Up

The first software step was to bring up the robot model and base control:

- `description.launch.py` started `robot_state_publisher`, `joint_state_publisher`, and `encoders_to_joint_state` with the URDF (`my_robot.urdf.xacro`), so the TF tree and wheel joints were available.

- `mecanum_bringup.launch.py` launched:

- `mecanum_controller` (I²C motor control + `wheel_encoders`),

- `mecanum_odometry` (`wheel_encoders` → `/odom` + TF `odom → base_link`),

- `joy` and `teleop_twist_joy` (PS5 mapping from `ps5.yaml`).

Using Foxglove, the team checked that wheels rotated correctly in the 3D model, TF frames (`odom`, `base_link`, `laser_frame`, wheels) were present, and `/odom` tracks roughly matched manual motions.

The TPR measurement tool was run to estimate ticks per revolution from encoder deltas over a single wheel rotation. The measured TPR was then set in the controller and odometry params, and straight-line motions over known distances were used to fine-tune wheel scales and signs.

5.3.4 Mapping and Map Saving

With odometry and LiDAR publishing, the mapping mode was enabled via the mode manager. `slam_toolbox` ran in online mapping mode, consuming `/scan` and `/odom` and publishing `/map` and `map → odom`. Using the PS5 controller or the Setup page joystick, the robot was driven around the environment to cover walls, aisles, and obstacles.

When the map looked clean and complete in Foxglove and on the Setup page, the operator entered a map name (e.g. `floor_a`) and clicked “Save Map”. The dashboard called `/save_map`; `state_manager` executed `map_saver_cli`, writing `floor_a.yaml` and the corresponding image into the maps directory. `/list_maps` confirmed that the map was available for navigation.

5.3.5 *Parking Spot Definition*

After switching to navigation mode with the saved map, Nav2 (AMCL + planner + controller) took over. The robot was either navigated or manually driven to the exact pose where the charging pad should sit relative to an EV for a given slot.

At that pose, the operator:

- Verified alignment visually and in Foxglove (pose in `/map`).

- Entered a label such as “B1-Row3-Spot7” in the Setup page.

- Clicked “Save Parking”.

The dashboard called `/save_parking_spot`; `state_manager` stored the map name, label, and pose in `<map_name>_parking.json`. Repeating this created a set of named parking/charging positions.

5.3.6 April Tag Docking Experiments

In parallel, vision-based docking was prototyped:

- The camera was calibrated with a chessboard using `camera_calibration.py`, producing `camera_matrix` and distortion coefficients (saved to `camera_params.npz`).

- An AprilTag (family `tag36h11`) of known physical size was generated via `tag_generator.py` and printed at 1:1 scale.

- The `april_tag_nav.py` script detected the tag, estimated its pose with `estimatePoseSingleMarkers`, and computed simple v, ω commands to correct distance and lateral offset.

With the robot stationary or moving slowly, these commands were observed and later tied into `cmd_vel`. Tests showed the camera could reliably provide the fine adjustments needed to bring the pad into a small alignment window around the tag.

5.3.7 *End-to-End Prototype Test*

Finally, all pieces were run together:

1. `bringup_all.launch.py` started URDF, mecanum control/odometry, and LiDAR.

2. The integration launch started `state_manager`, `mode_manager`, `rosbridge_server`, and the web dashboard.

3. Mapping mode produced a map that was saved and listed.

4. Navigation mode loaded the map; selecting a parking spot from the Dashboard triggered `/navigate_to_goal`. Nav2 drove the robot to the stored pose.

5. Near the parking pose, the docking logic (when enabled) refined alignment using the AprilTag view.

6. Charger telemetry from the EV-side ESP32 was received via ESP-NOW, forwarded to ROS, and displayed as live voltage/current/power on the Dashboard.

This step-by-step assembly and integration produced a working prototype that could map a parking area, remember charging positions, navigate autonomously to them, refine alignment visually, and expose its state and charging behavior through a modern web interface.

6 Chapter 6: Testing and Results

6.1 Testing Methodology (How the system was tested – unit testing, system testing, stress testing, etc.)

This section explains how the autonomous wireless-charging robot was tested. Testing was carried out in stages to make sure all components work correctly on their own, then together, and finally as a full system in different environments and conditions, not only in ideal situations.

6.1.1 Robot and chassis

We first verified that the Raspberry Pi communicates properly with the motor driver over I²C. We tested this by sending movement commands and confirming that the motors respond immediately and accurately. Wheel rotation direction, speed control, and odometry updates were checked continuously.

Each wheel was also tested individually to measure how many encoder ticks correspond to exactly one full revolution. We repeated this several times and took the average. These tick values are essential for accurate odometry, which later improves SLAM localization.

6.1.2 LiDar

We confirmed that the LiDAR was mounted at the correct height so that no part of the chassis or wheels blocks its view. The sensor was tested for stable 360-degree scanning with no blind spots. We also verified correct TF frame alignment so that /scan data appears in the right position in the coordinate system and is usable by SLAM and Nav2.

6.1.3 Camera

The camera was tested for its ability to detect QR codes and AprilTags of different sizes and at different distances. We checked its ability to determine tag position, color recognition, and approximate distance estimation. This step ensures the camera can later perform fine docking adjustments.

6.1.4 ESP-NOW

ESP-NOW communication was tested between the two ESP32 modules to confirm that power-related data (voltage, current, and power) can be transmitted reliably and received by the RPi. Tests were done with minimal Wi-Fi interference to confirm stable packet delivery and consistent updates on the screen.

6.1.5 Node execution and termination

We tested how ROS2 nodes start, stop, and interact with each other. This includes launching nodes in different modes (mapping or navigation) and ensuring they terminate cleanly without leaving background processes active.

6.2 Experimental Setup (Testbed description, measurement tools used, testing conditions)

All tests were conducted in controlled indoor environments that approximate typical parking-facility conditions, while still being safe for development.

The primary testbed consisted of:

- An open area of approximately 5 x 5 m with movable obstacles (cardboard boxes, chairs, tables) representing cars and pillars.
- Flat surfaces including ceramic tiles, painted concrete, and a small carpeted section to observe traction and odometry behaviour.
- A "charging bay" mock-up: a designated region where a printed AprilTag and, for some tests, a low-voltage charger emulator was placed to simulate an EV parking spot.

The robot was connected to a Wi-Fi network shared with the operator laptop running:

- Foxglove Studio for ROS visualization.
- A browser accessing the web dashboard via the Node.js server.
- SSH terminals for monitoring logs and launching ROS nodes.

For mapping and navigation, the LiDAR was run at its typical scan rate, and SLAM/Nav2 used the parameter files described earlier (map resolution 0.05 m, AMCL particle ranges, Nav2 controller limits, etc.). Sensor topics (/scan, /odom, /map) and transforms were inspected frequently to confirm timing and data quality.

For docking tests, the AprilTag was mounted at a fixed height representing a tag near an EV's receiver plate or on nearby infrastructure (such as a small stand). The camera was calibrated beforehand and connected directly to the Pi 5, with test scripts run locally.

ESP-NOW tests used the two ESP32 modules within indoor range (a few meters) with minimal Wi-Fi traffic to isolate protocol behavior. Later tests were performed with more devices on the network to see whether moderate interference affected packet delivery.

6.3 Results (Graphs, tables, numerical analysis of test results)

Because this prototype focuses on feasibility and system integration rather than industrial-grade performance, results emphasize key functional metrics: mapping quality, navigation accuracy, docking precision, and telemetry behavior.

6.3.1 Mapping Results

Using SLAM Toolbox, the robot successfully produced consistent 2D occupancy maps of the test area. Visual inspection in Foxglove and the dashboard showed:

- Walls and large obstacles aligned well with ground-truth measurements (tape-measure distances) within approximately 3–5 cm over a 5 m span.
- Loop closures occurred when the robot revisited previously explored regions; global map drift was reduced and corridors aligned without visible “kinks”.
- Small, movable obstacles (e.g., chairs) appeared as clusters of occupied cells and did not prevent mapping of the background structure.

Saved maps were re-loaded in navigation sessions and remained consistent; no corruption or major misalignment was observed between mapping and navigation runs.

6.3.2 Navigation and Localization Performance

With AMCL and Nav2 running on a saved map, the robot was able to navigate to goal poses across the test environment. Observed behaviors included:

- **Goal accuracy**: the robot typically stopped within about 5 cm of the requested goal position and within about 0.1 rad of the goal orientation, matching the configured goal tolerances in Nav2.
- **Obstacle avoidance**: when temporary obstacles (boxes) were placed in the planned path after the goal had been sent, the local planner selected alternative trajectories around them, provided sufficient clearance existed.
- **Relocalization**: after minor disturbances (e.g., nudging the robot slightly), AMCL recovered the pose estimate within a few seconds as new scans were incorporated.

Performance was slightly affected by floor surface; on carpet, odometry drift increased due to higher slip, but AMCL and the LiDAR map were able to compensate as long as scan quality was good.

6.3.3 Docking Precision

Initial docking experiments used the AprilTag detection scripts to compute tag pose and generate simple velocity commands. From starting positions approximately 0.3–0.5 m away and up to ±0.1 m laterally offset, repeated trials showed that:

- The camera reliably detected the tag when it was within the central part of the field of view and not heavily occluded.
- Estimated distance and lateral offset were stable enough to drive gradual corrections without oscillations when gains were chosen conservatively.

- Final lateral misalignment between the robot centerline and the tag could be reduced to on the order of 10–20 mm in most tests, consistent with the target of ≤ 2 cm.

These results are based on camera-only docking in a simplified environment. Integration into the full ROS 2 docking flow (and under the vehicle's underside geometry) remains future work, but the experiments confirm the viability of AprilTag-based fine alignment.

6.3.4 ESP-NOW Telemetry

In bench tests with simulated loads, the EV-side ESP32 sent voltage and current samples at rates between 5–10 Hz. The robot-side ESP32 and bridge node decoded and published these as ROS topics. Observations included:

- Negligible message latency relative to the sampling period.
- No visible packet drops in low-interference conditions over several minutes.
- Reported voltages and currents matched multimeter readings to within sensor tolerances.

In slightly noisier environments (other Wi-Fi traffic present), occasional frame loss occurred, but this manifested as brief gaps in the plotted telemetry and did not destabilize the overall system. For charging monitoring (not control), this behavior is acceptable.

6.4 Performance Evaluation (Was the project successful? , Comparison with design requirements and specifications)

Based on the above tests, the prototype can be evaluated against its main functional goals.

- **Mapping and localization**:
  The robot is able to create occupancy grid maps of medium-sized indoor spaces and then localize itself robustly within those maps. SLAM Toolbox and AMCL parameters can be tuned to trade off between responsiveness and noise tolerance. The mapping and localization performance is sufficient to feed Nav2 and to support semantic parking spot definitions.

- **Navigation**:
  Nav2, combined with the mecanum base and costmaps, provides smooth motion with reasonable goal accuracy and reliable obstacle avoidance in structured environments. Navigation remains robust across small layout changes and different surfaces, as long as the LiDAR line-of-sight is maintained.

- **Docking and alignment**:
  Camera-based docking using AprilTag pose estimation can achieve the fine alignment step that pure

LiDAR-based navigation alone cannot guarantee. Early results demonstrate the ability to reduce lateral misalignment to the order of 2 cm, which is compatible with typical WPT coil alignment requirements.

- **Telemetry and monitoring**:
  ESP-NOW telemetry integrated into ROS and presented via the dashboard offers real-time visibility of charging voltage, current, and power. Combined with /robot_state, /map, and /odom, the system gives operators a comprehensive view of robot and charger behavior.

- **Robustness to mode switching**:
  The mode manager cleanly switches between mapping and navigation without requiring manual restarts. This supports realistic workflows in which a facility might be mapped initially, then operated in navigation mode most of the time, and re-mapped only after significant layout changes.

Overall, while this is a prototype rather than a production system, the testing shows that the key architectural choices—LiDAR SLAM, encoder odometry, Nav2, AprilTag docking, and ESP-NOW telemetry—work together to meet the core objectives of autonomous, precise, and observable wireless charging alignment.

6.5 Limitations & Challenges (Issues encountered and their impact on results)

Several limitations were observed during testing:

- **Coverage and scale**:
  Tests were performed in a relatively small indoor area. Scaling to full-size parking decks introduces challenges such as longer paths, more reflective surfaces, and more dynamic obstacles.

- **Surface-dependent odometry**:
  On slippery or uneven surfaces, wheel slip increases odometry error. While SLAM/AMCL can correct this, convergence may be slower in low-feature zones or when obstacles are sparse.

- **Camera constraints**:
  Tag detection depends on lighting, occlusion, and the relative geometry between the camera, tag, and vehicle body. Dirty tags, strong reflections, or low light can degrade pose estimates.

- **Incomplete integration of docking into Nav2**:
  For this prototype, docking is validated primarily through test scripts and conceptual integration. A full integration with Nav2's docking server and behavior trees, as well as tests under actual vehicles, remains future work.

6.6 Improvements & Optimization (Proposed refinements for future work)

Future work can build on these results in several directions:

- **Full docking node integration**:
  Wrap the AprilTag detection and control logic into a dedicated ROS 2 node and integrate it with Nav2’s docking server so that docking becomes a first-class navigation behavior.
- **Extended environment testing**:
  Evaluate performance in larger, multi-zone maps, including multi-map operation and map switching, as would be found in multi-level parking structures.
- **Sensor fusion**:
  Combine LiDAR, camera, and possibly IMU data for more robust localization, especially in feature-poor or visually degraded areas.
- **Adaptive behavior**:
  Improve behavior trees to better handle dynamic conditions: adjusting speed based on crowding, handling temporarily blocked spots, or pausing missions when ESP-NOW telemetry indicates charger faults.
- **Hardware refinement**:
  Improve cable management and shielding to reduce EMI, experiment with different camera/lens configurations for wider field of view, and refine the pad mounting for easier alignment and maintenance.

By addressing these points, the prototype can evolve into a more robust and scalable system suitable for real-world deployment in large parking facilities.

7 Chapter 7: Conclusion & Future Work

7.1 Project Summary (Key takeaways from design, testing, and implementation)

This project developed a working prototype of an autonomous wireless EV charging robot designed for parking facilities, such as malls and commercial garages. Rather than building a wireless charger from scratch, the focus was on the autonomous alignment, navigation, and system integration required to make commercial wireless chargers practical and efficient.

On the hardware side, the robot combines a mecanum-wheel base with integrated encoders, a 2D LiDAR (YDLIDAR G4), a front-mounted camera, an off-the-shelf wireless charging transmitter pad, and two ESP32 modules for charger telemetry via ESP-NOW. Mechanically, the hardware was mounted to match a well-defined URDF description, so that the TF tree (map → odom → base_link → sensors) accurately reflects reality.

On the software side, the system is built on ROS 2 Jazzy and follows a layered architecture:

- A base control layer (mecanum_controller, mecanum_odometry, encoders_to_joint_state) converts cmd_vel into motor commands over I^{2}C and reconstructs odometry from encoder ticks.
- A perception/localization layer uses SLAM Toolbox to build occupancy maps and AMCL/Map Server to localize in saved maps, while maintaining the map → odom transform.
- The Nav2 stack (planners, controllers, costmaps, behavior trees) handles global and local navigation tasks.
- A camera-based docking layer, based on AprilTags and camera calibration, provides the final centimeter-level alignment corrections.
- A mission management layer (state_manager, mode_manager) supervises mapping vs. navigation modes, manages maps and parking spots, and publishes a unified /robot_state.
- A web dashboard and rosbridge interface provide an operator-friendly UI for mapping, mission dispatch, and monitoring, while Foxglove Studio supports engineering visualization.

Testing was carried out in stages: first unit tests of the base, LiDAR, camera, and ESP-NOW links; then integration tests of SLAM + odometry, Nav2 + AMCL, and camera docking; and finally full end-to-end tests in a small indoor environment. Results show that:

- The robot can map new environments with SLAM, save and reload maps, and localize reliably with AMCL on those maps.
- Nav2 can plan and execute collision-free paths in these maps, reaching goals with acceptable position and orientation accuracy.

- AprilTag-based docking can refine the final alignment to within ≈ 2 cm lateral error, suitable for high-efficiency inductive charging.
- ESP-NOW telemetry provides stable low-latency voltage and current measurements from the charger to the ROS system.

In summary, the project demonstrates a complete workflow from mapping and semantic parking-spot definition to autonomous navigation and fine docking, supported by a modern browser-based operator interface and ROS-native monitoring.

7.2 Contributions & Achievements (How the project met its objectives and industry relevance)

The primary contribution of this work is an end-to-end prototype that connects mapping, semantic parking-spot definition, autonomous navigation, precise docking, and charger telemetry into a coherent workflow. The system demonstrates that a mobile robot can map a parking zone once, store named charging locations within that map, and later navigate autonomously to those locations while compensating for moderate variations in starting pose and obstacles. The addition of camera-based docking closes the final alignment gap that LiDAR-only navigation cannot guarantee, pushing the residual misalignment into a range that is compatible with high-efficiency inductive power transfer.

At an architectural level, the project delivers a reusable ROS 2 design for autonomous operation in parking facilities. The separation into distinct packages for base control, robot description and mission management, message and service definitions, and camera docking creates a structure that can be adapted or extended for similar platforms. The mode manager concept, which starts and stops SLAM or Nav2 as external processes under supervision, offers a practical approach to multi-mode operation in real deployments, where mapping and navigation do not necessarily run at the same time. The use of a web dashboard, backed by rosbridge and a Node.js server, provides a modern and accessible operator interface without tying the system to specific hardware displays or native applications.

From an application perspective, the prototype addresses a real barrier in wireless EV charging: the sensitivity of inductive systems to misalignment. By automating pad positioning with centimeter-level accuracy, the system reduces dependence on driver precision and can improve both convenience and energy efficiency. It is particularly relevant for environments where manual plug-in is inconvenient or infeasible, such as for users with reduced mobility or in high-turnover parking facilities where staff would otherwise intervene. The work also illustrates a practical path toward “charging as a service” in future smart parking infrastructures, where robots handle the physical connection between grid and vehicle.

7.3 Future Enhancements (How the project can be improved or extended)

Although the prototype meets its main technical objectives, it remains a research platform rather than a product, and there are several clear directions for future improvement.

One important area is the integration and hardening of the docking behavior. At present, AprilTag-based docking has been validated in scripts and early tests, but it is not yet embedded as a full Nav2 docking action with robust failure handling. Implementing a dedicated docking node that publishes or consumes poses in a standardized way, then integrating it through Nav2's docking server or behavior trees, would make docking a first-class navigation primitive. This should be followed by extensive testing under actual vehicle geometries and in more challenging visual conditions, including low light, reflections, and partial tag occlusion.

Scaling to larger and more complex environments is another important step. The current tests have been carried out in relatively small, single-level areas. Real parking facilities have multiple rows, ramps, and levels, often with stronger reflections and more dynamic obstacles. Extending the system to support multi-map operation at building scale, with efficient map selection and localization on each level, would move the system closer to practical deployment. This may involve more aggressive optimization of SLAM parameters, stronger relocalization strategies, or even the use of semantic landmarks in addition to pure geometry.

Sensor fusion is a natural extension point. The current system relies on wheel odometry, LiDAR, and camera data, but fusion is limited to LiDAR–odometry SLAM and LiDAR–map AMCL. Combining IMU measurements, visual odometry, and LiDAR in a more unified framework could improve robustness in low-feature environments or on surfaces with poor traction. At the same time, better integration of charger telemetry into decision-making could evolve the system from pure monitoring to active management, for example by automatically detecting charging completion, flagging anomalies, or adjusting missions based on power availability.

Finally, there is room to refine both hardware and human–robot interaction. Mechanically, the robot could benefit from a more compact and rugged chassis, improved cable management, and modular sensor mounts that simplify adjustment and maintenance. On the user side, the dashboard could be extended to support multiple robots, role-based access, and integration with parking and fleet-management systems. These enhancements would make the solution more attractive for real operators and provide a path for scaling from a single proof-of-concept robot to a fleet operating in a live facility.

In conclusion, the project demonstrates that combining mature robotics technologies—LiDAR-based SLAM, probabilistic localization, Nav2 planning, fiducial-based visual docking, and lightweight wireless telemetry—can yield a practical autonomous charging assistant. With further work on docking integration, large-scale deployment, sensor fusion, and interface refinement, this approach has the potential to become a realistic option for supporting EV charging in the next generation of smart parking infrastructures.

8 References (Use IEEE referencing style)

[1]	International Energy Agency (IEA), “Global EV Outlook 2023,” 2023. Available: https://www.iea.org/reports/global-ev-outlook-2023
[2]	V. Ravikiran, “Review on contactless power transfer for electric vehicle charging,” Energies, vol. 10, no. 5, p. 636, May 2017. https://www.mdpi.com/1996-1073/10/5/636.
[3]	G. Palani, U. Sengamalai, P. Vishnuram, and B. Nastasi, “Challenges and Barriers of Wireless Charging Technologies for Electric Vehicles,” Energies, vol. 16, no. 5, p. 2138, Feb. 2023, doi: https://doi.org/10.3390/en16052138
[4]	“16.1 Maxwell’s Equations and Electromagnetic Waves - University Physics Volume 2 | OpenStax,” openstax.org. https://openstax.org/books/university-physics-volume-2/pages/16-1-maxwells-equations-and-electromagnetic-waves
[5]	P. Jayathurathnage, A. Alphones, and D. Vilathgamuwa, “Coil optimization against misalignment for wireless power transfer,” in Proc. Int. Conf. Electr. Eng. Informat., 2017, pp. 1–6. https://www.researchgate.net/publication/313538899_Coil_optimization_against_misalignment_for_wireless_power_transfer.
[6]	Bumper Charger. (n.d.). Car Charging Robot. Retrieved February 2025, from https://bumpercharger.com
[7]	M. Talaat, I. Arafa, and H. M. B. Metwally, "Advanced automation system for charging electric vehicles based on machine vision and finite element method," IET Electric Power Applications, vol. 15, no. 1, pp. 1-9, Jan. 2021, doi: 10.1049/elp2.12038.
[8]	D. Fox, W. Burgard, and S. Thrun, "The Dynamic Window Approach to Collision Avoidance," IEEE Robotics & Automation Magazine, vol. 4, no. 1, pp. 23-33, Mar. 1997
[9]	Z. Wei, S. Wang, K. Chen, and F. Wang, "ROS-Based Navigation and Obstacle Avoidance: A Study of Architectures, Methods, and Trends," Sensors, vol. 25, no. 14, p. 4306, Jul. 2025. doi: 10.3390/s25144306.
[10]	“Setting Up Transformations — Nav2 1.0.0 documentation,” *Nav2 Docs*, 2024. docs.nav2.org
[11]	H. Taheri, B. Qi ao and N. Ghaeminezhad, “Kinematic Model of a Four-Mecanum-Wheeled Mobile Robot,” Int. J. Computer Applications, vol. 113, no. 3, Mar. 2015.
[12]	“ROS2 tf2 Tutorial - Coordinate Frame Transform,” YouTube video, 2023. https://www.youtube.com/watch?v=CraslJJkrcU
[13]	“ROS 2 Introduction,” Husarion Tutorials, 2024. https://husarion.com/software
[14]	YDLIDAR G4 — Hardware Manual and Range Specifications, 2023. https://cdn.robotshop.com/media/y/ydl/rb-ydl-03/pdf/ydlidar-g4-datasheet-1.pdf
[15]	S. Macenski et al., “SLAM Toolbox: SLAM for the Real World,” Open Source Robotics Foundation, 2020. https://www.researchgate.net/publication/351568967_SLAM_Toolbox_SLAM_for_the_dynamic_world
[16]	S. Kohlbrecher et al., “A Flexible and Scalable SLAM System with ROS,” RoboCup Symposium, 2011. https://www.researchgate.net/publication/228852006_A_flexible_and_scalable_SLAM_system_with_full_3D_motion_estimation
[17]	“navigation - ROS Wiki,” wiki.ros.org. https://wiki.ros.org/navigation

[18]	"amcl - ROS Wiki," wiki.ros.org. https://wiki.ros.org/amcl
[19]	"Nav2 - ROS 2 Navigation Stack — ros-docs documentation," Neobotix-docs.de, 2023. https://neobotix-docs.de/ros/ros2/autonomous_navigation.html
[20]	ROBOMECHTRIX, "Amcl | ROS Localization | SLAM 2 | How to localize a robot in ROS | ROS Tutorial for Beginners," YouTube, Feb. 03, 2021. https://www.youtube.com/watch?v=ZfQ30rfJb08 (accessed Nov. 22, 2025)
[21]	[5] Why, "Why navfn is using Dijkstra?," Robotics Stack Exchange, Feb. 23, 2012. https://robotics.stackexchange.com/questions/38174/why-navfn-is-using-dijkstra (accessed Nov. 22, 2025).
[22]	jidianwsj, "espressif-docs-readthedocs-hosted-com-arduino-esp32-en-latest," Scribd, 2025. https://www.scribd.com/document/751078541/espressif-docs-readthedocs-hosted-com-arduino-esp32-en-latest (accessed Nov. 22, 2025).
[23]	A. Triviño, J. M. González-González, and J. A. Aguado, "Wireless Power Transfer Technologies Applied to Electric Vehicles: A Review," Energies, vol. 14, no. 6, p. 1547, Mar. 2021, doi: https://doi.org/10.3390/en14061547.
[24]	OpenCV, "OpenCV: Camera Calibration," docs.opencv.org. https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html
[25]	E. Olson, "AprilTag: A robust and flexible visual fiducial system," IEEE Xplore, May 01, 2011. https://ieeexplore.ieee.org/document/5979561
[26]	United Nations, "The 17 sustainable development goals," United Nations, 2015. https://sdgs.un.org/goals
[27]	United Nations, "The 17 sustainable development goals," United Nations, 2015. https://sdgs.un.org/goals
[28]	"SAE J1772," Wikipedia, Dec. 13, 2021. https://en.wikipedia.org/wiki/SAE_J1772
[29]	[13] "INTERNATIONAL STANDARD NORME INTERNATIONALE Electric vehicle wireless power transfer (WPT) systems - Part 2: Specific requirements for MF-WPT system communication and activities Systèmes de transfert de puissance sans fil (WPT) pour véhicules électriques - Partie 2: Exigences spécifiques pour la communication et les activités des systèmes MF-WPT." Available: https://cdn.standards.iteh.ai/samples/103762/a586973b18574edf8d44fd2f63dcaf8c/IEC-61980-2-2023.pdf
[30]	IEC, "IEC 61010-1:2010," Webstore.iec.ch, 2016. https://webstore.iec.ch/en/publication/4279
[31]	"ROS Package by License," Ros.org, 2025. https://docs.ros.org/en/diamondback/api/licenses.html (accessed Nov. 22, 2025).
[32]	"ISO/IEC/IEEE International Standard - Software and systems engineering –Software testing –Part 1:General concepts," ISO/IEC/IEEE 29119-1:2022(E), pp. 1–60, Jan. 2022, doi:

	https://doi.org/10.1109/IEEESTD.2022.9698145.

9 Appendices