Autonomous Solar Vehicle
Innovating Sustainable Transportation

Sardar Shahzeb Khan	**(BSE221059)**
Saqib Nawaz Khan	**(BSE223175)**
Shehriyar Ali Rustam	**(BSE223190)**

Supervised By

Syed Awais Haider

Fall 2025

BS Software Engineering

Department of Software Engineering

Capital University of Science & Technology, Islamabad

Project Report

VERSION	V 1.0	NUMBER OF MEMBERS	3

TITLE	Autonomous Solar Vehicle - Innovating Sustainable Transportation

SUPERVISOR NAME	SYED AWAIS HAIDER

MEMBER NAME	REG. NO.	EMAIL ADDRESS
SARDAR SHAHZEB KHAN	BSE221059	shahzeb3303@gmail.com
SAQIB NAWAZ KHAN	BSE223175	saqib055nawaz@gmail.com
SHEHRIYAR ALI RUSTAM	BSE223190	shehriyarali122@gmail.com

MEMBERS' SIGNATURES

Supervisor's Signature

Approval Certificate

This project, entitled as **Autonomous Solar Vehicle - Innovating Sustainable Transportation** has been approved for the award of

Bachelors of Science in Software Engineering

Committee Signatures:

Supervisor: ______________________________

(Mr. Syed Awais Haider)

Project Coordinator: ______________________________

(Mr. Ibrar Arshad)

Head of Department: ______________________________

(Dr. Nadeem Anjum)

Declaration

We, hereby, declare that "No portion of the work referred to, in this project has been submitted in support of an application for another degree or qualification of this or any other university/institute or other institution of learning". It is further declared that this undergraduate project, neither as a whole nor as a part thereof has been copied out from any sources, wherever references have been provided.

MEMBERS' SIGNATURES

Acknowledgements

The team would like to express heartfelt gratitude to the supervisor, **Mr. Syed Awais Haider**, for their invaluable guidance and support throughout the development of the Autonomous Solar Vehicle project. Their expertise and encouragement were instrumental in navigating challenges and enhancing understanding of sustainable technology. The team also extends thanks to peers and family for their unwavering support and motivation, which inspired them to push the boundaries of innovation in this endeavor.

Dedication

This project is dedicated to the team members’ parents, whose unwavering support and belief in their potential have inspired them to strive for excellence. Their dedication to hard work and perseverance serves as a constant motivation in the pursuit of success. Also, this effort is dedicated to the team mentors and friends, whose encouragement and guidance have pushed them to exceed limits and embrace challenges with confidence.

Executive Summary

The Autonomous Solar Vehicle project was initiated to address growing concerns about road safety, environmental pollution, rising fuel costs, and human error in driving. By integrating artificial intelligence, embedded sensors, and solar energy harvesting, the project demonstrates the viability of a self-driving, solar-powered vehicle as a sustainable and efficient mobility solution for controlled environments.

Through extensive research and development, a small-scale autonomous prototype was designed and tested, powered entirely by a 50-watt solar panel and managed through a Raspberry Pi 4 processing unit. Key features include ultrasonic sensor-based obstacle detection, camera-based object recognition using TensorFlow Lite, GPS-enabled autonomous navigation, and a cross-platform Flutter mobile application for real-time monitoring and control. The resulting prototype showcases significant benefits, including reduced dependency on fossil fuels, elimination of driver requirements, and enhanced operational efficiency in localized settings.

Beyond proving technical feasibility, this project highlights the transformative potential of combining renewable energy with autonomous navigation technology, paving the way for affordable, intelligent, and green transportation solutions particularly suited for university campuses, hospitals, industrial zones, and smart residential societies.

Table of Contents

List of Tables

List of Figures

This page is kept blank

Chapter 1

Introduction

Transportation is one of the most essential aspects of modern society, yet it continues to face challenges such as rising fuel costs, traffic congestion, environmental degradation, and dependency on human drivers. The need for safer, cleaner, and more efficient mobility solutions has never been more urgent. In response to these challenges, the integration of autonomous driving technologies with renewable energy systems presents a promising pathway toward sustainable transportation [1].

This project, titled "Autonomous Solar Vehicle - Innovating Sustainable Transportation," focuses on designing and developing a small-scale autonomous car prototype powered entirely by solar energy. Unlike conventional vehicles that rely on fossil fuels or grid-powered electricity, this vehicle demonstrates how renewable energy can be harnessed to operate intelligent transportation systems independently. The prototype is specifically designed for controlled environments such as university campuses, industrial zones, hospitals, and smart residential societies, where eco-friendly and automated transport solutions are increasingly valuable.

The proposed system integrates artificial intelligence, embedded hardware, and solar energy harvesting to achieve safe, reliable, and efficient navigation without human intervention. Hardware components such as Raspberry Pi 4, ultrasonic sensors, a GPS module, and a camera form the backbone of the prototype, enabling features like real-time object detection, obstacle avoidance, and autonomous route planning [2, 3]. A compact solar system powers the entire model, showcasing the feasibility of renewable-powered smart vehicles.

By combining autonomy with sustainability, this project not only addresses transportation efficiency and safety but also contributes toward reducing the carbon footprint of mobility systems. Ultimately, the Autonomous Solar Vehicle highlights the potential of affordable, intelligent, and green technologies in shaping the future of transportation, particularly in regions where cost-effective and sustainable solutions are needed most.

1.1 Existing Examples / Solutions

Autonomous driving has been an active field of research and commercial development for the past two decades. Several companies and research labs have introduced systems that aim to achieve full self-driving capabilities, though most solutions are still under testing and refinement [4].

Google's Waymo Project: One of the most recognized autonomous driving initiatives, Waymo uses advanced LiDAR, radar, and AI algorithms to navigate urban environments [5]. While highly advanced, its deployment is limited due to high costs, infrastructure requirements, and legal restrictions.

Tesla Autopilot & Full Self-Driving (FSD): Tesla's system relies heavily on computer vision and camera-based object detection [6]. Although it provides semi-autonomous features such as lane centering, adaptive cruise control, and auto lane changes, the system still requires driver supervision and has faced criticism regarding safety in complex scenarios.

Uber ATG & Other Startups: Companies like Uber ATG and Aurora have experimented with self-driving taxis, integrating AI with mapping technologies. However, most of these systems are costly, require extensive datasets, and are not easily adaptable for small-scale or localized implementations.

Academic Prototypes: Many universities worldwide have developed autonomous car prototypes using Raspberry Pi, Arduino, and sensors such as ultrasonic modules and cameras. While these systems successfully demonstrate navigation in controlled environments, they often lack robustness, adaptability, and affordability for local institutions in developing countries.

While these existing solutions demonstrate the potential of autonomous driving, they share common limitations that the Autonomous Solar Vehicle project aims to address. Commercial systems like Waymo and Tesla are prohibitively expensive and designed for public road environments, making them unsuitable for localized, controlled settings. Academic prototypes, though affordable, typically lack integration with renewable energy systems and rely on grid-charged batteries. The proposed Autonomous Solar Vehicle distinguishes itself by combining autonomous navigation capability with solar energy harvesting in a single low-cost prototype, specifically targeting controlled environments where these technologies are most immediately viable and beneficial.

1.2 Business Scope

This section outlines the business scope of the proposed project by presenting a Lean Canvas model. The table below summarizes the key business elements, including the problem statement, proposed solution, value proposition, target customers, and revenue structure. It provides a clear overview of how the project addresses existing transportation challenges while ensuring sustainability, cost efficiency, and long-term scalability.

Table 1.1: Lean Canvas Template

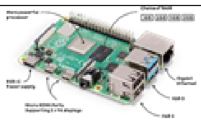
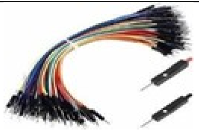
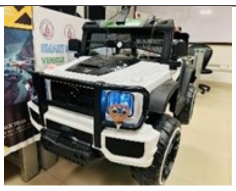
Problem	**Solution**	**Unique Value Proposition**	**Unfair Advantage**	**Customer Segments**
1. Traditional fuel-based vehicles are expensive to operate and contribute to environmental pollution. 2. Grid-powered electric vehicles still rely on non-renewable energy sources. 3. Dependence on human drivers increases operational costs and safety risks.	A self-driving vehicle powered primarily by solar energy that offers clean, low-cost, and independent transportation in localized environments such as campuses, hospitals, and industrial sites.	A sustainable and autonomous transportation solution that reduces fuel costs, eliminates driver dependency, and operates entirely on renewable solar energy providing both environmental and economic benefits.	The combination of solar power with autonomous navigation using affordable hardware (like Raspberry Pi) and open-source software makes it scalable, cost-effective, and difficult for competitors to replicate easily.	Universities, hospitals, industrial complexes, residential societies, and future smart cities looking for eco-friendly, efficient, and automated transport solutions.

Existing Alternatives	**Key Metrics**	**High-Level Concept**	**Channels**	**Early Adopters**
Fuel-based vehicles, electric shuttles relying on grid power, and manually operated campus transport systems.	Reduction in operational costs, energy efficiency (solar utilization rate), distance covered per charge, safety performance, and customer adoption rate.	An intelligent self-driving solar vehicle for sustainable transport.	Direct sales and partnerships with institutions, government smart city programs, technology exhibitions, and online awareness campaigns focusing on sustainability.	University campuses and industrial parks seeking eco-friendly, automated, and low-maintenance transport systems.

Cost Structure	Revenue Structure
Solar panels, sensors, Raspberry Pi units, batteries, motors, manufacturing, testing, software development, maintenance, and promotional costs.	Revenue from vehicle sales, leasing models for institutions, maintenance contracts, and possible integration partnerships for smart city transport networks.

1.3 Useful Tools and Technologies

Table 1.2: Hardware Components

Sr. No.	Components	Symbols/Model	Short Description
1	Ultrasonic sensor		Measures distance using sound waves
2	Camera		Captures images and videos for path planning and obstacle detection
3	Raspberry Pi 4 Model B		A single-board computer for running various applications and controlling devices
4	Connection Wires		Conduct electrical current between components
5	Solar plates		50 watt solar plate
6	Toy Car		Toy car for practical implementation of the prototype

1.4 Project Work Breakdown

This diagram presents a Work Breakdown Structure (WBS) for an Autonomous Solar Vehicle final year project. It hierarchically breaks the project into seven main phases: Project Initiation, Requirements Analysis, Design, Hardware Integration, Software Development, Testing, and Deployment. Each phase is further divided into specific tasks such as system architecture design, sensor integration, algorithm development, testing, and final deployment. The WBS provides a clear, organized view of the project scope, responsibilities, and deliverables, helping ensure effective planning, execution, and project management.

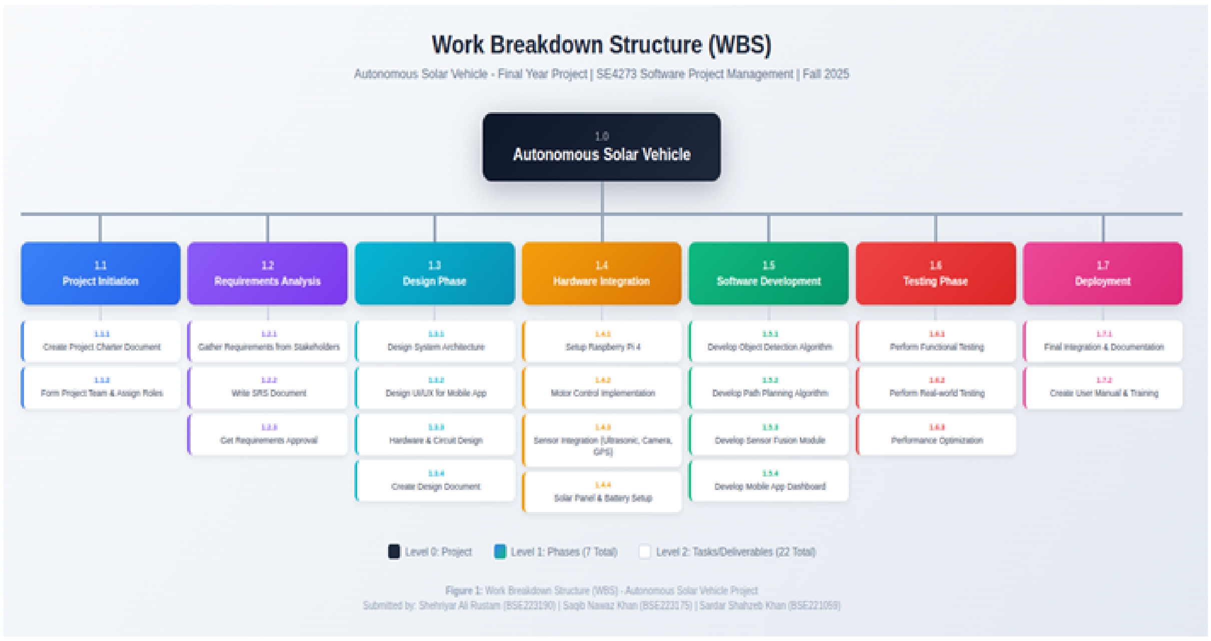

Work Breakdown Structure (WBS)
Autonomous Solar Vehicle - Final Year Project | SE4273 Software Project Management | Fall 2025
1.0 Autonomous Solar Vehicle
1.1 Project Initiation
1.1.1 Create Project Charter Document
1.1.2 Form Project Team & Assign Roles
1.2 Requirements Analysis
1.2.1 Gather Requirements from Stakeholders
1.2.2 Write SRS Document
1.2.3 Get Requirements Approval
1.3 Design Phase
1.3.1 Design System Architecture
1.3.2 Design UI/UX for Mobile App
1.3.3 Hardware & Circuit Design
1.3.4 Create Design Document
1.4 Hardware Integration
1.4.1 Setup Raspberry Pi 4
1.4.2 Motor Control Implementation
1.4.3 Sensor Integration (Ultrasonic, Camera, GPS)
1.4.4 Solar Panel & Battery Setup
1.5 Software Development
1.5.1 Develop Object Detection Algorithm
1.5.2 Develop Path Planning Algorithm
1.5.3 Develop Sensor Fusion Module
1.5.4 Develop Mobile App Dashboard
1.6 Testing Phase
1.6.1 Perform Functional Testing
1.6.2 Perform Real-world Testing
1.6.3 Performance Optimization
1.7 Deployment
1.7.1 Final Integration & Documentation
1.7.2 Create User Manual & Training
Level 0: Project
Level 1: Phases (7 Total)
Level 2: Tasks/Deliverables (22 Total)
Figure 1: Work Breakdown Structure (WBS) - Autonomous Solar Vehicle Project
Submitted by: Shehriyar Ali Rustam (BSE223190) | Saqib Nawaz Khan (BSE223175) | Sardar Shahzeb Khan (BSE221069)

Figure 1.1: Project Work Breakdown

1.5 Project Timeline

The development of the Autonomous Solar Vehicle is planned across two semesters, divided into structured phases that ensure steady progress from concept to implementation. Each phase has clearly defined goals to maintain focus and alignment with project objectives shown in Figure 1.2.

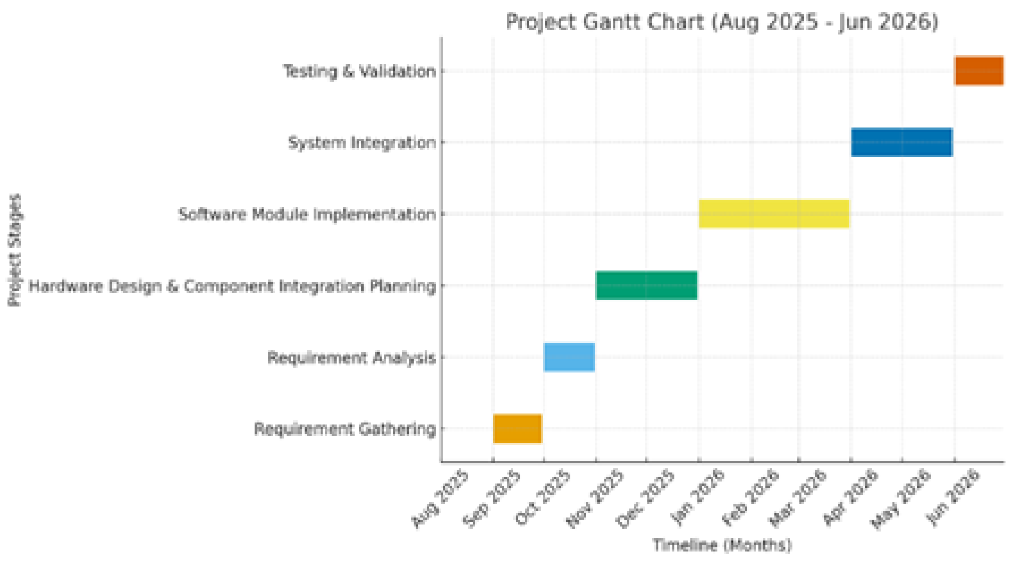


Figure 1.2: Project Timeline

1.6 Conclusion

Chapter 1 has presented the background and significance of developing autonomous solar-powered vehicles as a sustainable alternative to conventional transportation systems. The discussion outlined the challenges associated with fuel-based and grid-dependent vehicles, while emphasizing the potential of integrating renewable energy with intelligent navigation technologies. A review of existing solutions demonstrated advancements in autonomous driving; however, issues of cost, adaptability, and applicability in localized contexts remain evident. The proposed Autonomous Solar Vehicle seeks to address these limitations by offering a cost-effective and environmentally friendly model designed for controlled environments such as campuses, hospitals, and industrial zones. Furthermore, the scope of the project and its essential hardware components have been introduced, providing a comprehensive basis for the subsequent chapters that will detail its design, development, and implementation.

Chapter 2

Requirement Specification and Analysis

The development of the Autonomous Solar Vehicle requires a comprehensive understanding of both functional and non-functional requirements to ensure reliable, safe, and efficient performance in real-world scenarios. This chapter outlines the foundational requirements and analytical processes that guide the vehicle's design and behavior by identifying system goals, constraints, and user expectations, establishing a structured approach for transforming the project concept into a fully functional prototype. It begins with the definition of high-level Epics major system capabilities such as autonomous navigation, real-time obstacle detection, and solar energy management which are then decomposed into detailed User Stories describing system behaviors from the perspectives of various stakeholders, including passengers, operators, maintenance staff, and safety officers. These user stories clarify how the vehicle should interact with its environment, process incoming data, and respond to dynamic conditions. The analysis also incorporates acceptance criteria for each user story to ensure measurable and testable outcomes during system validation, maintaining alignment between project objectives and implementation [7, 8]. Overall, this chapter establishes the groundwork for subsequent stages of system design, testing, and implementation, ensuring that every component of the Autonomous Solar Vehicle including sensors, AI algorithms, solar energy systems, and navigation logic is developed according to clear, structured, and validated requirements.

2.1 Epics

2.1.1 Epic 1

Epic E1: Autonomous Navigation and Path Planning

Description: As a system administrator, I want the vehicle to navigate autonomously between specified locations within controlled environments so that it can transport passengers or goods without human intervention while optimizing routes for efficiency and safety.

2.1.2 Epic 2

Epic E2: Real-time Obstacle Detection and Avoidance

Description: As a safety officer, I want the vehicle to detect and avoid obstacles including pedestrians, other vehicles, and static objects in real-time so that it can operate safely in dynamic environments and prevent accidents.

2.1.3 Epic 3

Epic E3: Solar Energy Management and Optimization

Description: As an environmental officer, I want the vehicle to efficiently harvest, store, and manage solar energy so that it can operate sustainably with minimal dependency on grid power and reduce carbon emissions.

2.1.4 Epic 4

Epic E4: Remote Monitoring and Control

Description: Remote Monitoring and Control Description: As a fleet manager, I want to remotely monitor the vehicle's status, location, battery level, and sensor health through a mobile application so that I can ensure safe operation and intervene when necessary without being physically present at the vehicle.

2.2 User Stories

2.2.1 User Story 1

E1-US1: Destination Input and Route Planning

Description: As a passenger, I want to input my destination through a touchscreen interface so that the vehicle can automatically calculate and follow the optimal route to my destination.

Acceptance Criteria: Given the vehicle is in standby mode, When I select a destination from the available locations, Then the system should display the route, estimate time of arrival, and begin navigation upon confirmation.

2.2.2 User Story 2

E1-US2: GPS-based Position Tracking

Description: As a system operator, I want the vehicle to continuously track its position using GPS so that it can maintain accurate location awareness and follow planned routes.

Acceptance Criteria: Given the GPS module is active, When the vehicle is in motion, Then the system should update position coordinates every second with accuracy within 2 meters.

2.2.3 User Story 3

E2-US1: Pedestrian Detection

Description: As a pedestrian, I want the vehicle to detect my presence and stop at a safe distance so that I can cross paths without risk of collision.

Acceptance Criteria: Given the vehicle is in motion, when a pedestrian is detected within 5 meters, Then the vehicle should initiate braking and stop at least 2 meters from the pedestrian.

2.2.4 User Story 4

E2-US2: Dynamic Obstacle Avoidance

Description: As a maintenance technician, I want the vehicle to navigate around temporary obstacles so that operations continue even when the usual path is blocked.

Acceptance Criteria: Given an obstacle is detected on the planned path, when alternative routes exist, Then the vehicle should calculate and follow a safe detour within 5 seconds.

2.2.5 User Story 5

E2-US3: Collision Warning System

Description: As a safety inspector, I want audible or visual warnings when obstacles are detected.

Acceptance Criteria: Given an obstacle is detected, When the vehicle initiates avoidance maneuvers, then warning lights should flash and an alert sound should activate within 200 milliseconds.

2.2.6 User Story 6

E3-US1: Solar Panel Efficiency Monitoring

Description: As a vehicle operator, I want the solar panels to collect and deliver power to the battery system so that the vehicle can operate using renewable energy.

Acceptance Criteria: Given solar panels are exposed to sunlight, When the vehicle is operational or parked, Then the solar panels should continuously harvest available solar energy and charge the battery system.

2.2.7 User Story 7

E4-US1: Real-time Vehicle Status Monitoring

Description: As a fleet manager, I want to view the vehicle's real-time status including battery level, GPS position, speed, and sensor health through the mobile application so that I can monitor operations remotely.

Acceptance Criteria: Given the mobile app is connected to the vehicle via WiFi, When the dashboard screen is opened, Then the app should display live battery percentage, current speed, GPS coordinates, and sensor status updated every 2 seconds.

2.3 Test Cases

2.3.1 Test Case 1: Destination Input and Route Calculation

Test case for verifying destination input and route calculation functionality for E1-US1 (Destination Input and Route Planning).

Table 2.1: Destination Input Test Case

Test Case ID	E1-US1-TC1
Test Case Description	Verify that the touchscreen interface correctly accepts destination input and calculates optimal routes
Input	Preconditions: Vehicle in standby mode, touchscreen active. Input: Select "Engineering Building Block A" from destination list. User Action: Press "Confirm" button
Expected Result	1. Route displayed on map screen. 2. Estimated time of arrival shown (e.g., "5 minutes"). 3. Total distance displayed (e.g., "1.2 km"). 4. "Start Navigation" button appears. 5. Navigation begins after confirmation

2.3.2 Test Case 2: Invalid Destination Input

Negative test case for destination input validation for E1-US1.

Table 2.2: Invalid Destination Input Test Case

Test Case ID	E1-US1-TC2
Test Case Description	Verify system handles invalid or out-of-range destinations appropriately
Input	Preconditions: Vehicle in standby mode. Input: Attempt to enter destination outside campus boundary. User Action: Type "City Mall" (outside operational area)
Expected Result	1. Error message: "Destination outside operational area". 2. Suggestion to select from available locations. 3. List of valid destinations displayed. 4. System remains in standby mode

2.3.3 Test Case 3: GPS Position Tracking

Test case for GPS-based position tracking functionality for E1-US2.

Table 2.3: GPS Position Tracking Test Case

Test Case ID	E1-US2-TC1
Test Case Description	Verify GPS module provides accurate and timely position updates during vehicle movement
Input	Preconditions: GPS module initialized, clear sky view. Vehicle State: Moving at 10 km/h. Test Duration: 60 seconds. Test Route: Predefined 500m circular path
Expected Result	1. Position updates received every 1 second. 2. Accuracy within 2 meters of actual position. 3. No gaps in position data. 4. Coordinates format: Latitude/Longitude to 6 decimal places. 5. Speed calculation matches actual speed ±1 km/h

2.3.4 Test Case 4: Pedestrian Detection

Test case for pedestrian detection and emergency braking for E2-US1.

Table 2.4: Pedestrian Detection Test Case

Test Case ID	E2-US1-TC1
Test Case Description	Verify ultrasonic sensors detect pedestrians and vehicle stops at safe distance
Input	Preconditions: All sensors operational, clear weather. Vehicle Speed: 10 km/h. Test Mannequin: Placed 5 meters ahead. Sensor Range: Active (ultrasonic + camera). Test Surface: Flat paved road
Expected Result	1. Pedestrian detected at 5 meters distance. 2. Alert triggered immediately. 3. Automatic braking initiated. 4. Vehicle stops 2-3 meters from pedestrian. 5. Warning sound activated. 6. Hazard lights flash

2.3.5 Test Case 5: Dynamic Obstacle Avoidance

Test case for dynamic obstacle avoidance functionality for E2-US2.

Table 2.5: Dynamic Obstacle Avoidance Test Case

Test Case ID	E2-US2-TC1
Test Case Description	Verify vehicle can detect and navigate around temporary obstacles on its path
Input	Preconditions: Vehicle following planned route. Obstacle: Construction cone placed on path. Vehicle Speed: 10 km/h. Available Space: 3 meters on left, 2 meters on right. Obstacle Size: 1m x 1m
Expected Result	1. Obstacle detected at 4+ meters distance. 2. Alternative path calculated within 5 seconds. 3. Turn signal activated (left). 4. Vehicle navigates around obstacle. 5. Returns to original path after passing. 6. Total detour time < 15 seconds

2.3.6 Test Case 6: Complete Path Blockage

Test case for complete path blockage handling for E2-US2.

Table 2.6: Complete Path Blockage Test Case

Test Case ID	E2-US2-TC2
Test Case Description	Verify system behavior when path is completely blocked
Input	Preconditions: Vehicle on narrow path. Scenario: Large obstacle blocking entire width. No Alternative Path: Walls on both sides. Vehicle Speed: 8 km/h. Distance to obstacle: 10 meters
Expected Result	1. Obstacle detected. 2. System scans for alternatives (3 seconds). 3. Determines no path available. 4. Vehicle stops 3 meters from obstacle. 5. Alert: "Path blocked, requesting assistance". 6. Waits for manual intervention or obstacle removal

2.3.7 Test Case 7: Collision Warning System

Test case for collision warning system for E2-US3.

Table 2.7: Collision Warning System Test Case

Test Case ID	E2-US3-TC1
Test Case Description	Verify audible and visual warnings activate when obstacles are detected
Input	Preconditions: Warning system armed. Vehicle Speed: 12 km/h. Obstacle Type: Static object (trash can). Distance: 4 meters ahead. Ambient Conditions: Daylight, quiet environment
Expected Result	1. Obstacle detected at 4 meters. 2. Warning lights flash within 200ms. 3. Alert sound activates (65-75 dB). 4. Pattern: 2 beeps per second. 5. LED lights: Amber flashing. 6. Warnings stop after obstacle passed

2.3.8 Test Case 8: Solar Charging While Driving

Test case for solar charging while vehicle is in motion for E3-US1.

Table 2.8: Solar Charging While Driving Test Case

Test Case ID	E3-US1-TC3
Test Case Description	Verify solar panels continue charging battery while vehicle is moving
Input	Preconditions: Sunny conditions. Vehicle State: In motion at 15 km/h. Route: Open area with sun exposure. Battery Level: 60%. Power Consumption: Normal operation. Test Duration: 30 minutes
Expected Result	1. Solar panels generate power while moving. 2. Power output: 25-35W (considering angle changes). 3. Net positive charging when consumption < generation. 4. No interference with vehicle operation. 5. Smooth power management. 6. Battery level maintained or slightly increased

2.4 User Interface Design (Prototypes)

User Interface (UI) Design focuses on anticipating what users might need to do and ensuring that the interface has elements that are easy to access, understand, and use to facilitate those actions. UI brings together concepts from interaction design, visual design, and information architecture. The following sections describe the UI design of each page in the Solar Drive autonomous vehicle mobile application.

In this application, the UI is designed with an emphasis on clarity, consistency, and user safety. Visual elements are arranged to minimize cognitive load while still presenting all critical information in an organized and meaningful way. Standard design patterns are applied across screens to ensure a familiar experience, allowing users to navigate the application intuitively with minimal learning effort. Attention is also given to responsiveness and accessibility, ensuring that the interface performs smoothly across different devices and remains usable in varying lighting and driving conditions. Overall, the UI design aims to enhance user confidence and interaction efficiency while supporting the advanced functionalities of the Solar Drive autonomous vehicle system.

Main Dashboard screen as shown in Figure 2.1 will contain three circular gauge widgets for displaying battery, speed, and solar input. It includes four metric cards showing power statistics and five navigation tab buttons at the bottom. The screen also contains multiple label elements displaying average speed, max speed, and range values.

The layout of the Main Dashboard is designed to present critical information in a clear and organized manner, enabling users to monitor system performance at a glance. The circular gauges provide an intuitive visual representation of changing values, while the metric cards offer precise numerical data for deeper insight into power usage and efficiency. The navigation tab buttons ensure smooth and consistent movement between different screens of the application.

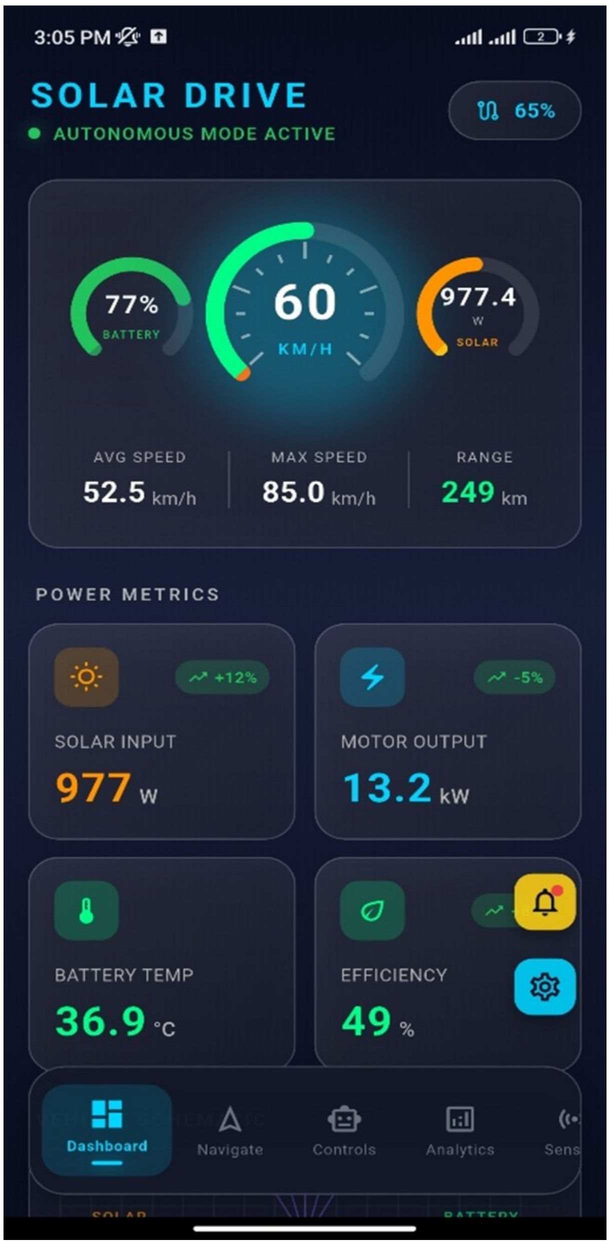


Figure 2.1: Main Dashboard Screen

Vehicle Schematic screen as shown in Figure 2.2 will contain one vehicle schematic image view at the center with four corner label displays. It includes four list items in System Health section, each containing one icon, two text labels, and one status badge. The bottom navigation bar contains five tab buttons.

This screen is designed to provide a visual and diagnostic overview of the vehicle's condition. The central schematic image helps users quickly understand the physical layout of the vehicle, while the corner labels display key status indicators related to major subsystems. The System Health list presents detailed information in a structured format, allowing users to easily identify the operational state of critical components through icons, descriptive text, and color-coded status badges.

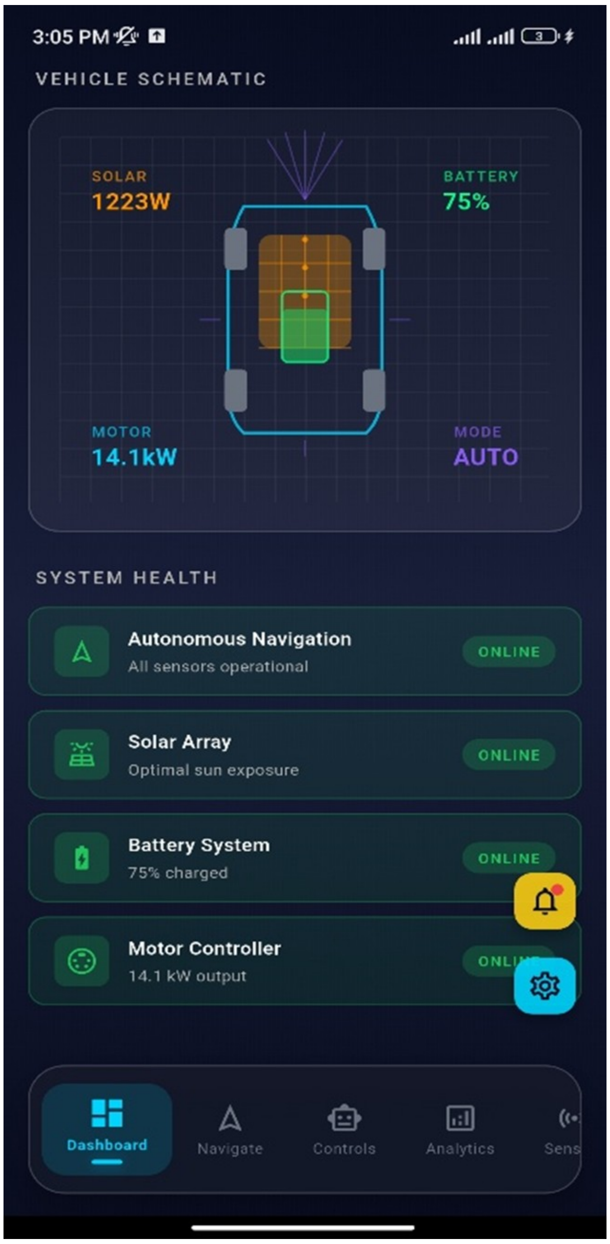


Figure 2.2: Vehicle Schematic Screen

Navigation screen as shown in Figure 2.3 will contain two text input fields for pickup and destination locations. It includes one voice input button, one edit button, and one primary action button for starting the ride. The screen contains one map view component with three zoom control buttons and three route option selection buttons.

In addition to these elements, the Navigation screen is designed to support ease of use and efficient route planning. The text input fields allow users to manually enter locations, while the voice input button provides a hands-free alternative for improved convenience and safety. The edit button enables quick modification of entered locations without restarting the process, and the primary action button clearly indicates the transition from planning to navigation.

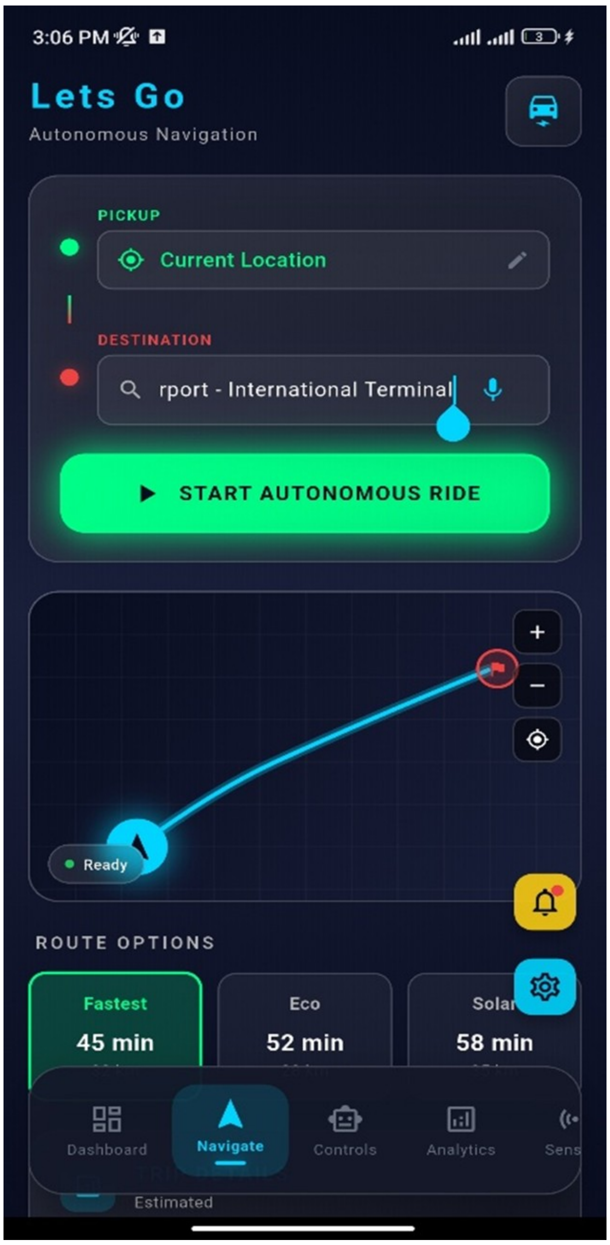


Figure 2.3: Navigation Screen

Control Panel screen as shown in Figure 2.4 will contain one large circular power button for autonomous mode activation. It includes four toggle switches for Lane Assist, Collision Avoidance, Adaptive Cruise, and Auto Parking features. Each toggle item contains one icon and two text labels for title and description. The screen is designed with a clear and minimal layout to ensure safe and intuitive interaction while enabling advanced driving features. The prominent placement of the circular power button emphasizes its importance and allows users to easily enable or disable autonomous mode.

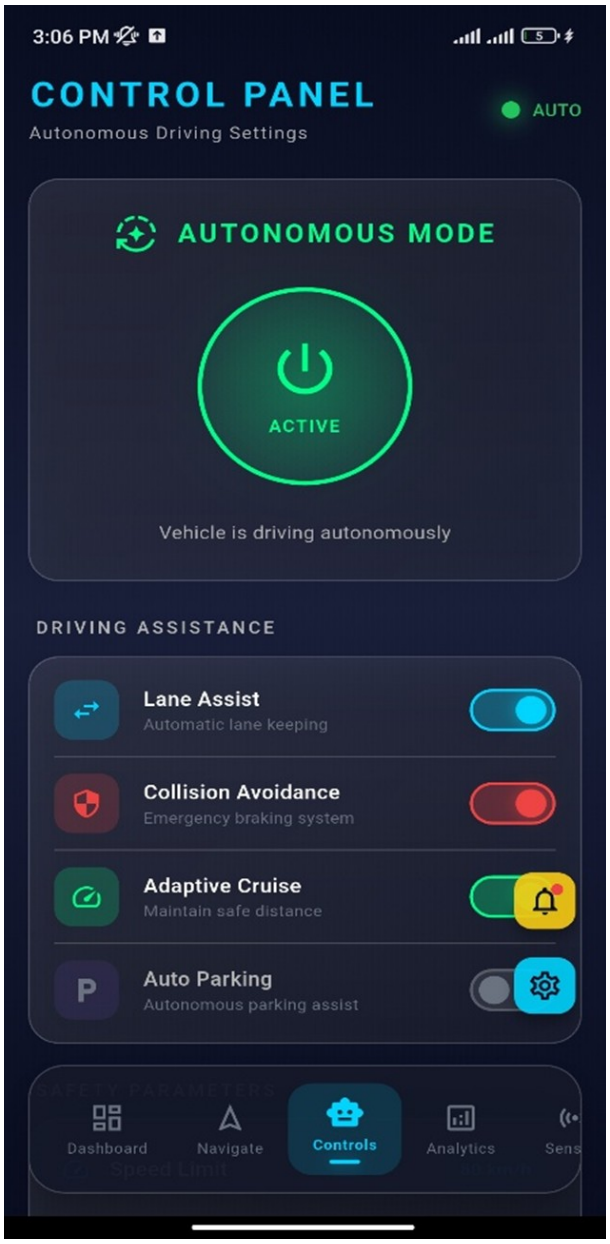
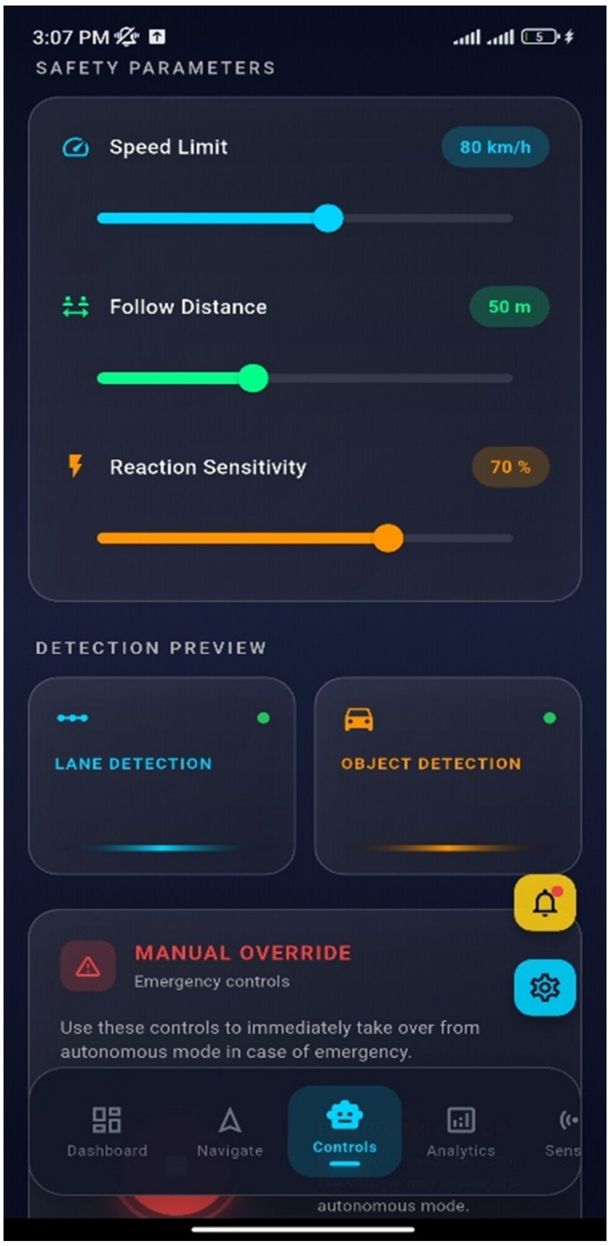


Figure 2.4: Control Panel Screen

Safety Parameters screen as shown in Figure 2.5 will contain three slider controls for Speed Limit, Follow Distance, and Reaction Sensitivity settings. It includes two detection preview cards with status indicators and one warning card for Manual Override section. Each slider displays one value badge showing current selection.

This screen is designed to allow users to fine-tune safety-related behaviors of the autonomous vehicle according to their preferences and driving conditions. The slider controls provide an intuitive way to adjust parameters with immediate visual feedback through the value badges. The detection preview cards give users real-time insight into the status of key sensing and detection systems, helping to build trust in the vehicle's awareness of its surroundings.


3:07 PM

SAFETY PARAMETERS

Speed Limit 80 km/h

Follow Distance 50 m

Reaction Sensitivity 70 %

DETECTION PREVIEW

LANE DETECTION

OBJECT DETECTION

MANUAL OVERRIDE

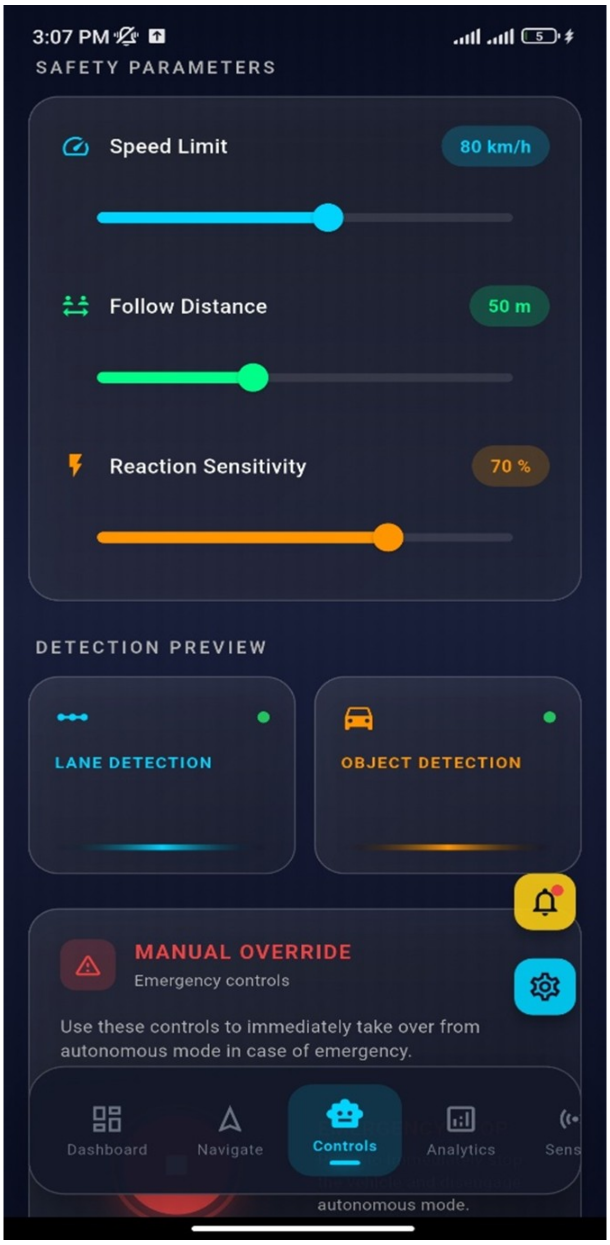
Emergency controls

Use these controls to immediately take over from autonomous mode in case of emergency.

Figure 2.5: Safety Parameters Screen

Safety Parameters Extended View as shown in Figure 2.6 will contain three slider controls same as Figure 2.5 with full visibility. It includes two detection preview cards with animated scan line indicators. The screen contains two floating action buttons for notifications and settings, and one warning card with descriptive text.

This extended view is designed to give users a more detailed and interactive interface for monitoring and configuring safety parameters. The fully visible sliders allow precise adjustments of Speed Limit, Follow Distance, and Reaction Sensitivity, while the animated scan lines in the detection cards offer an intuitive visualization of the vehicle's real-time sensing capabilities.



Use these controls to immediately take over from autonomous mode in case of emergency.

Figure 2.6: Safety Parameters Extended View

Analytics Solar Tab screen as shown in Figure 2.7 will contain three time filter buttons and three tab segment buttons. It includes two metric cards displaying output values and one line chart component for solar generation. The Panel Efficiency section contains three circular progress indicator widgets.

This screen is designed to give users a detailed and interactive overview of the vehicle's solar energy performance. The time filter buttons allow quick selection of data ranges, while the tab segments enable switching between different analytical perspectives. Metric cards present essential numerical insights at a glance, and the line chart helps users track changes in solar generation over time, highlighting patterns or anomalies.

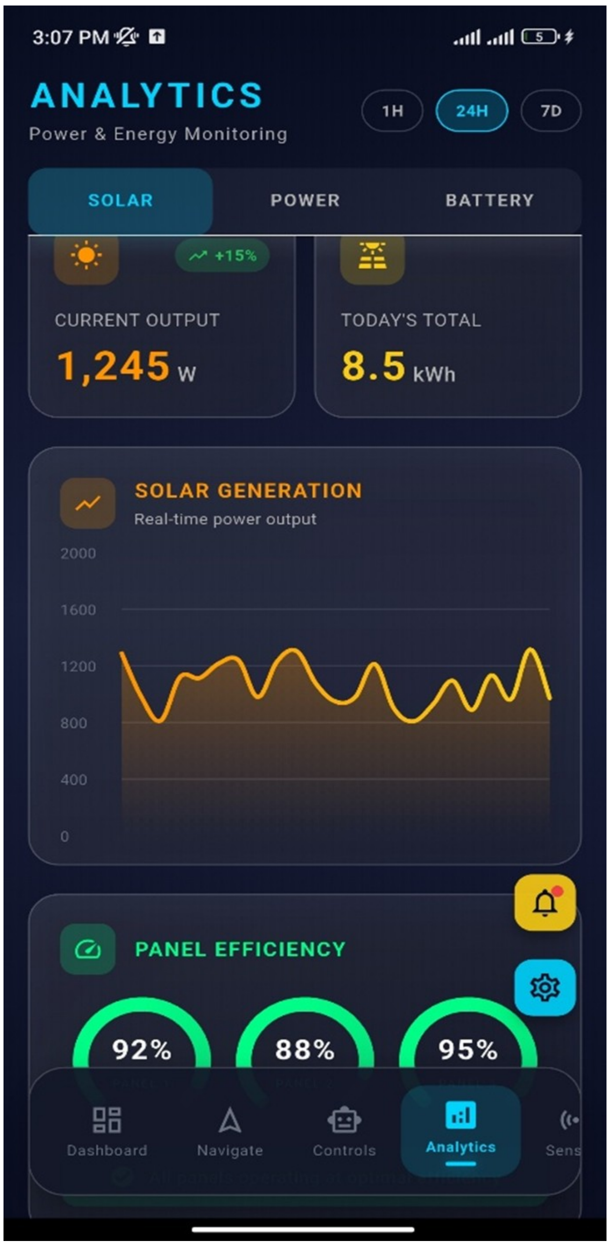


Figure 2.7: Analytics Solar Tab

Analytics Power Tab screen as shown in Figure 2.8 will contain two metric cards for motor output and efficiency display. It includes one line chart component showing power consumption over time. The screen contains two floating action buttons and one section header for Power Distribution.

This screen is designed to provide users with a comprehensive view of the vehicle's power performance. The metric cards offer quick, at-a-glance insights into motor output and efficiency, helping users understand real-time performance. The line chart presents a clear visual representation of power consumption patterns, enabling users to identify trends or potential issues.

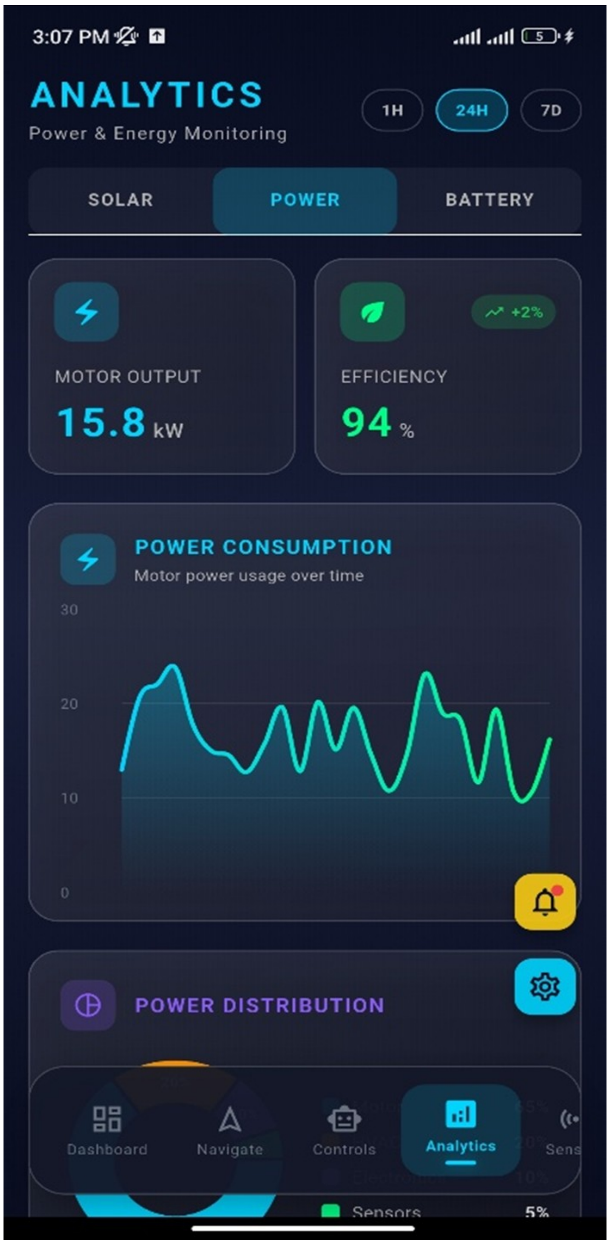


Figure 2.8: Analytics Power Tab

Analytics Battery Tab screen as shown in Figure 2.9 will contain two metric cards for charge level and remaining range. It includes one area chart component displaying battery level trends. The Battery Health section contains three label displays for health, cycles, and temperature values.

This screen is designed to give users a detailed understanding of the vehicle's battery status and performance. The metric cards provide quick access to critical numerical information, while the area chart helps track changes in battery level over time, making it easier to identify usage patterns or potential issues. The Battery Health section offers a concise summary of the battery's condition, including its overall health, number of charge-discharge cycles, and temperature, supporting proactive maintenance and efficient energy management.

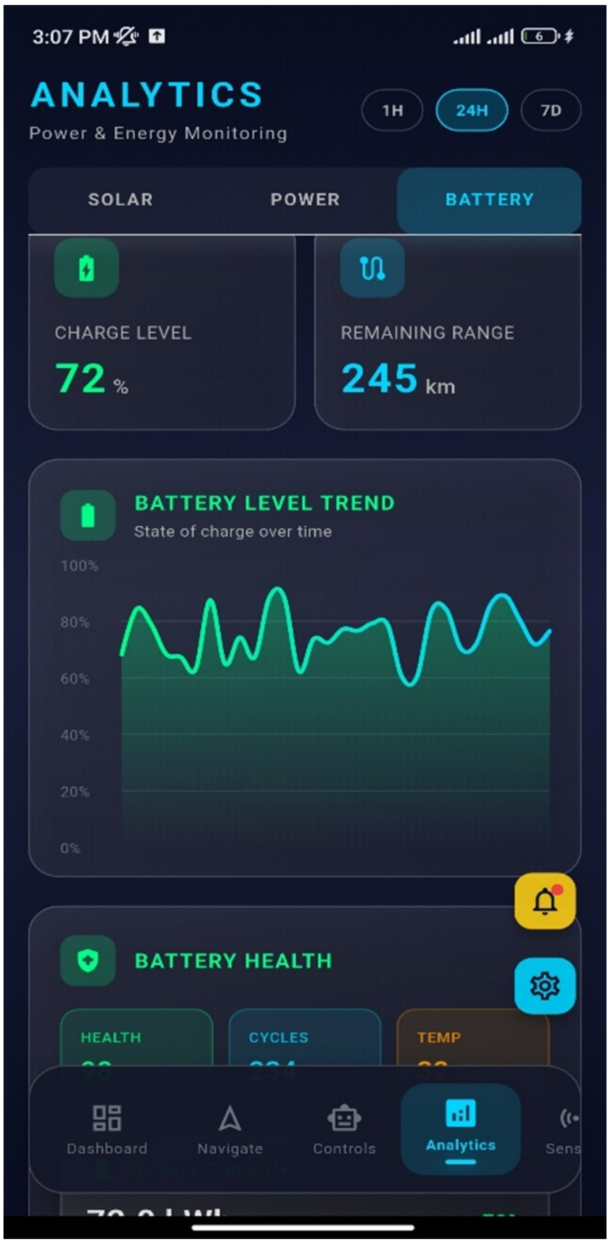


Figure 2.9: Analytics Battery Tab

Sensors Overview screen as shown in Figure 2.10 will contain four sensor status cards arranged in a 2x2 grid layout. It includes one video player component with four camera angle tab buttons. The screen contains one status badge in header and one LIDAR preview card at the bottom.

This screen is designed to give users a comprehensive and interactive view of the vehicle's sensor systems. The grid of sensor cards allows for at-a-glance monitoring of multiple sensor statuses simultaneously. The video player with camera angle tabs provides real-time visual feedback from various perspectives, enhancing situational awareness. The header status badge offers a quick summary of overall sensor functionality, while the LIDAR preview card presents detailed scanning information for advanced environmental perception.

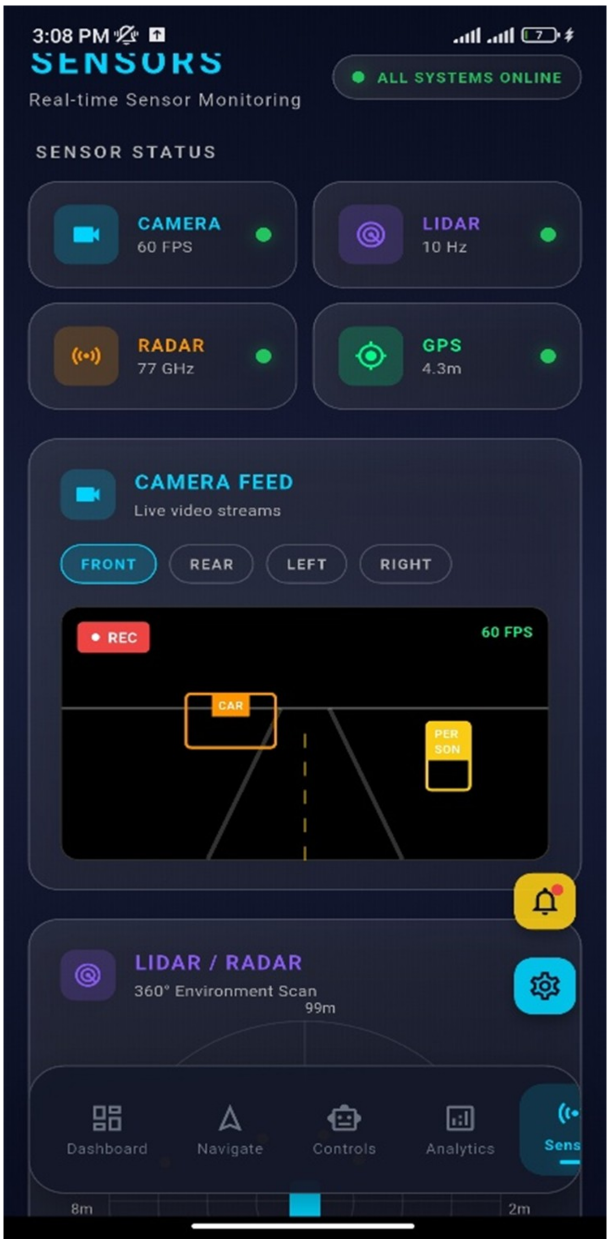


Figure 2.10: Sensors Overview Screen

Sensors LIDAR Detail screen as shown in Figure 2.11 will contain one circular radar display component showing 360-degree scan. It includes four distance label displays arranged in cross layout for front, rear, left, and right proximity. The screen contains one vehicle icon at center and one section header for GPS Location.

This screen is designed to provide users with an in-depth view of the vehicle's surrounding environment. The circular radar display gives a comprehensive 360-degree visualization of nearby objects, enhancing situational awareness. The distance labels clearly show proximity measurements in each direction, allowing users to quickly assess potential obstacles.

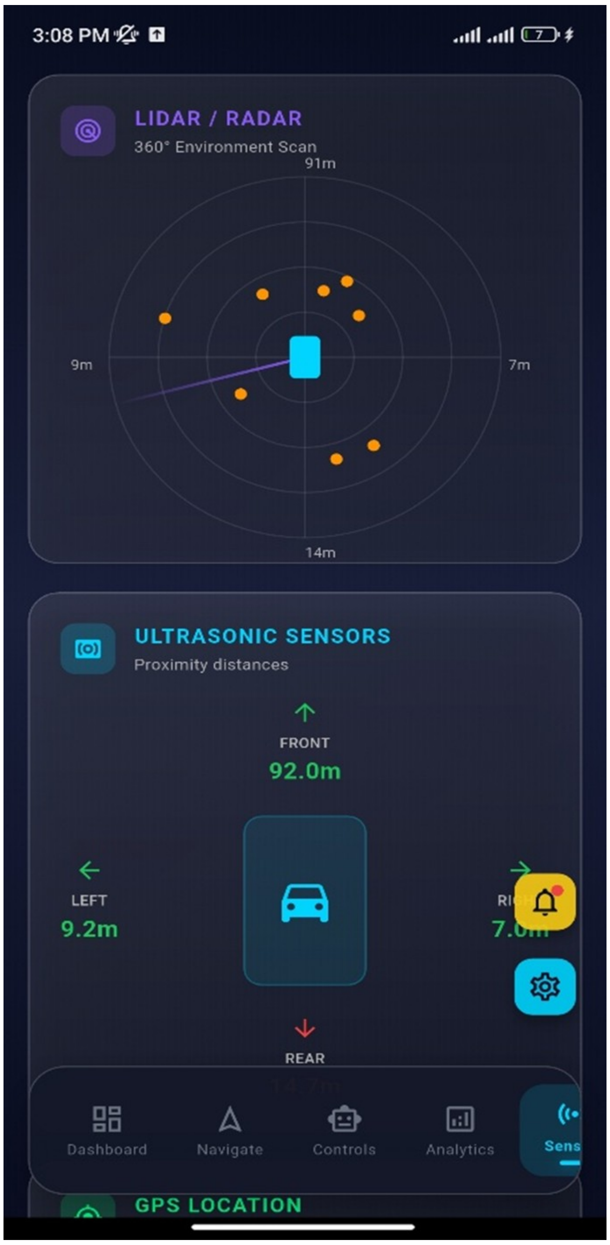
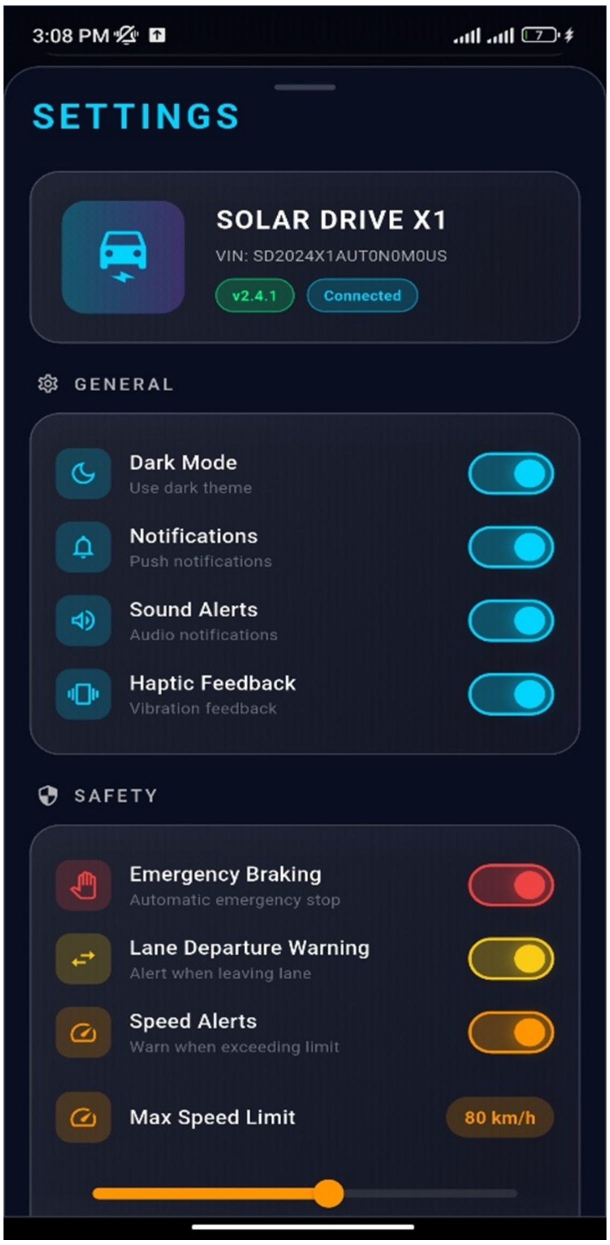


Figure 2.11: Sensors LIDAR Detail Screen

Settings Screen as shown in Figure 2.12 will contain one vehicle info card with two status badges. It includes four toggle switches in General section and three toggle switches in Safety section. The screen contains one slider control for max speed limit setting and multiple icon-label pairs for each setting item.

This screen is designed to offer users centralized control over both general and safety-related vehicle settings. The vehicle info card presents important information at a glance, while the toggle switches allow for quick adjustments of frequently used features. The slider provides precise control over the maximum speed limit, and the icon-label pairs ensure that each setting is clearly identified and easy to understand.


3:08 PM

SETTINGS

SOLAR DRIVE X1
VIN: SD2024X1AUTONOMOUS
v2.4.1 Connected

GENERAL

Dark Mode
Use dark theme

Notifications
Push notifications

Sound Alerts
Audio notifications

Haptic Feedback
Vibration feedback

SAFETY

Emergency Braking
Automatic emergency stop

Lane Departure Warning
Alert when leaving lane

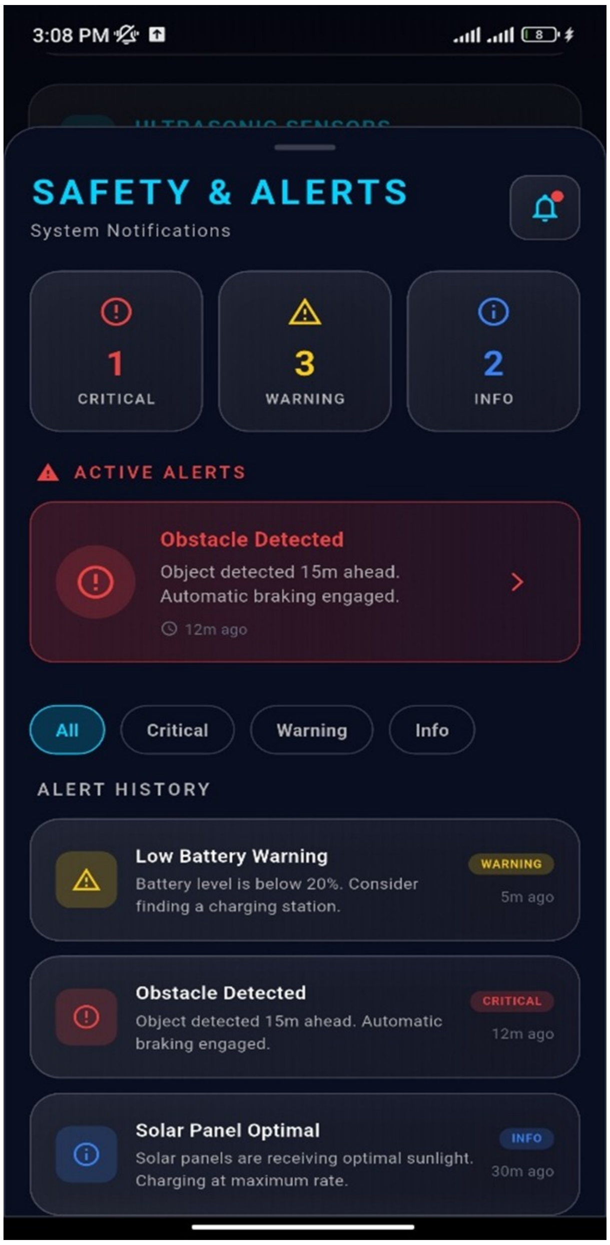
Speed Alerts
Warn when exceeding limit

Max Speed Limit 80 km/h

Figure 2.12: Settings Screen

Safety Alerts screen as shown in Figure 2.13 will contain three summary cards displaying alert counts by severity level. It includes four filter tab buttons and one expandable active alert card. The Alert History section contains multiple list items, each with one icon, two text labels, one badge, and one timestamp.

This screen is designed to give users a clear and organized view of the vehicle's safety notifications. The summary cards offer a high-level overview of alert distribution, allowing users to quickly identify critical concerns. Filter tab buttons enable efficient navigation through different types of alerts, while the expandable active alert card provides in-depth details for immediate attention.



SAFETY & ALERTS

System Notifications

1 CRITICAL

3 WARNING

2 INFO

ACTIVE ALERTS

Obstacle Detected

Object detected 15m ahead. Automatic braking engaged.

12m ago

All | Critical | Warning | Info

ALERT HISTORY

Low Battery Warning — WARNING — 5m ago

Battery level is below 20%. Consider finding a charging station.

Obstacle Detected — CRITICAL — 12m ago

Object detected 15m ahead. Automatic braking engaged.

Solar Panel Optimal — INFO — 30m ago

Solar panels are receiving optimal sunlight. Charging at maximum rate.

Figure 2.13: Safety Alerts Screen

2.5 Traceability Matrix

The Traceability Matrix in Table 2.9 provides a structured mapping between high-level epics, their associated user stories, and corresponding test cases. It ensures that all functional requirements of the Solar Drive autonomous vehicle system are accounted for and can be systematically verified. Each Epic ID represents a major feature area, while the User Story IDs break these features down into specific, testable requirements.

Table 2.9: Traceability Matrix

Epic ID	**Epic Name**	**User Story ID**	**User Story Name**	**Test Case ID(s)**	**Priority**
E1	Autonomous Navigation and Path Planning	E1-US1	Destination Input and Route Planning	TC-E1-US1-01, TC-E1-US1-02	High
E1	Autonomous Navigation and Path Planning	E1-US2	GPS-based Position Tracking	TC-E1-US2-01	High
E2	Real-time Obstacle Detection and Avoidance	E2-US1	Pedestrian Detection	TC-E2-US1-01	Critical
E2	Real-time Obstacle Detection and Avoidance	E2-US2	Dynamic Obstacle Avoidance	TC-E2-US2-01, TC-E2-US2-02	High
E2	Real-time Obstacle Detection and Avoidance	E2-US3	Collision Warning System	TC-E2-US3-01	High
E3	Solar Energy Management and Optimization	E3-US1	Solar Panel Efficiency Monitoring	TC-E3-US1-03	Medium
E4	Remote Monitoring and Control	Not Implemented	-	-	Future
E5	Safety and Emergency Systems	Not Implemented	-	-	Future

2.6 Conclusion

The requirement specification and analysis phase provided a structured foundation for understanding the operational needs, system behavior, and performance expectations of the Autonomous Solar Vehicle. By defining comprehensive Epics, breaking them down into actionable User Stories, and attaching clear acceptance criteria, this chapter ensures that every system feature aligns with stakeholder expectations and project goals. Through this analytical process, both functional and non-functional requirements were identified, including safety, reliability, energy efficiency, and user interaction needs, which collectively guide the development team during the design and implementation phases and serve as measurable benchmarks for testing and validation. Overall, this chapter establishes a clear roadmap for the system's development, ensuring that the Autonomous Solar Vehicle is built on well-understood, thoroughly analyzed, and logically structured requirements, allowing the project to progress confidently toward system modeling, architectural design, simulation, and implementation.

Chapter 3

System Design

The system design chapter provides the blueprint of the Autonomous Solar Vehicle, bridging the gap between the requirements specified in Chapter 2 and the actual implementation. This chapter outlines how different components of the system are structured, interact with each other, and support the required functionality. The goal is to present a clear and organized design that ensures all user requirements are properly addressed and can be effectively translated into development. The Autonomous Solar Vehicle integrates multiple subsystems, including solar energy harvesting, autonomous navigation, obstacle detection, and motor control, all working together to achieve sustainable and intelligent transportation.

3.1 System Architecture (Layer Architecture)

The Autonomous Solar Vehicle employs a layered architecture that organizes the system into distinct functional layers, each responsible for specific tasks. This architectural approach was selected because it provides a clear separation of concerns, making the system easier to develop, test, and maintain [9]. The layered design also allows for independent modification of individual layers without affecting others, which is crucial for an embedded system that combines hardware sensors, processing units, and software algorithms. The system architecture is organized into five primary layers: Energy Layer, Sensing Layer, Processing Layer, Control Layer, and User Layer as shown in Figure 3.1. The Energy Layer handles solar power harvesting and battery management [10, 11], ensuring continuous power supply to all components of the system. The Sensing Layer comprises GPS modules [12], cameras, and ultrasonic sensors [13] that collect environmental data. The Processing Layer, centered around the Raspberry Pi 4, executes the main control logic, machine learning algorithms for object detection, and path planning computations. The Control Layer manages motor drivers [14] and steering mechanisms for vehicle movement. Finally, the User Layer provides interfaces including a mobile application, LCD display, and status LEDs for user interaction and system monitoring.

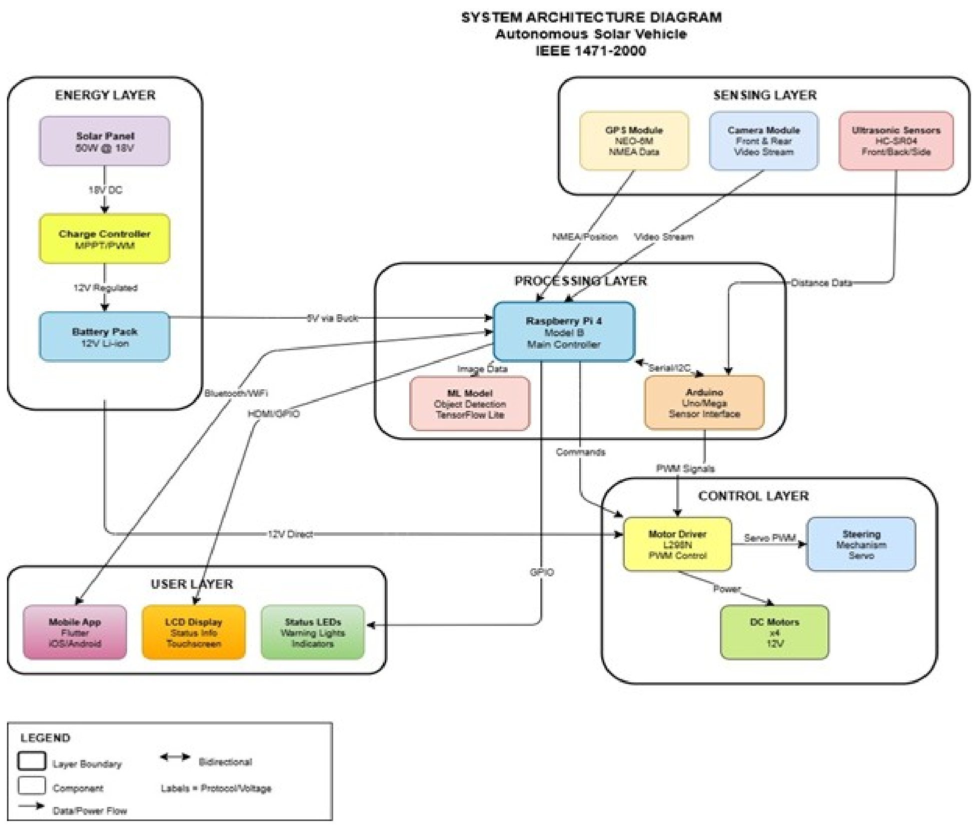

SYSTEM ARCHITECTURE DIAGRAM
Autonomous Solar Vehicle
IEEE 1471-2000
ENERGY LAYER
Solar Panel 50W @ 18V
18V DC
Charge Controller MPPT/PWM
12V Regulated
Battery Pack 12V Li-ion
SENSING LAYER
GPS Module NEO-6M NMEA Data
Camera Module Front & Rear Video Stream
Ultrasonic Sensors HC-SR04 Front/Back/Side
NMEA/Position
Video Stream
Distance Data
PROCESSING LAYER
5V via Buck
Raspberry Pi 4 Model B Main Controller
Image Data
ML Model Object Detection TensorFlow Lite
Serial/I2C
Arduino Uno/Mega Sensor Interface
Bluetooth/WiFi
HDMI/GPIO
Commands
PWM Signals
CONTROL LAYER
12V Direct
Motor Driver L298N PWM Control
Servo PWM
Steering Mechanism Servo
Power
DC Motors x4 12V
GPIO
USER LAYER
Mobile App Flutter iOS/Android
LCD Display Status Info Touchscreen
Status LEDs Warning Lights Indicators
LEGEND
Layer Boundary
Component
Data/Power Flow
Bidirectional
Labels = Protocol/Voltage

Figure 3.1: System Architecture Diagram

3.2 Components and Connector Diagram

The Autonomous Solar Vehicle system integrates four distinct component categories that work together to achieve autonomous, solar-powered transportation. Understanding how these components connect and communicate is essential for system integration and troubleshooting. The Hardware Components subsystem includes power generation and storage units (Solar Panel 50W, Charge Controller, Battery Pack 12V), main processing units (Raspberry Pi 4, Arduino Mega), sensing modules (GPS Module, Camera Module, Ultrasonic Array) and actuation components (Motor Driver L298N, DC Motors). Each component exposes specific interfaces: the Solar Panel provides IDCPower, the Raspberry Pi offers IProcessing, INavigation, and ISensorData interfaces, while the Motor Driver provides IMotorCtrl and IPWM interfaces. The component diagram as shown in Figure 3.2 presents the UML 2.5 standard view showing both hardware and software components. The diagram is divided into two subsystems: Hardware Components showing the physical devices and their power/data connections, and Software Components showing the ML Object Detection, Path Planner, Sensor Fusion, and Mobile App modules. The Raspberry Pi 4 serves as the central hub, delegating sensor interface tasks to Arduino Mega while handling all processing, navigation, and ML inference tasks.

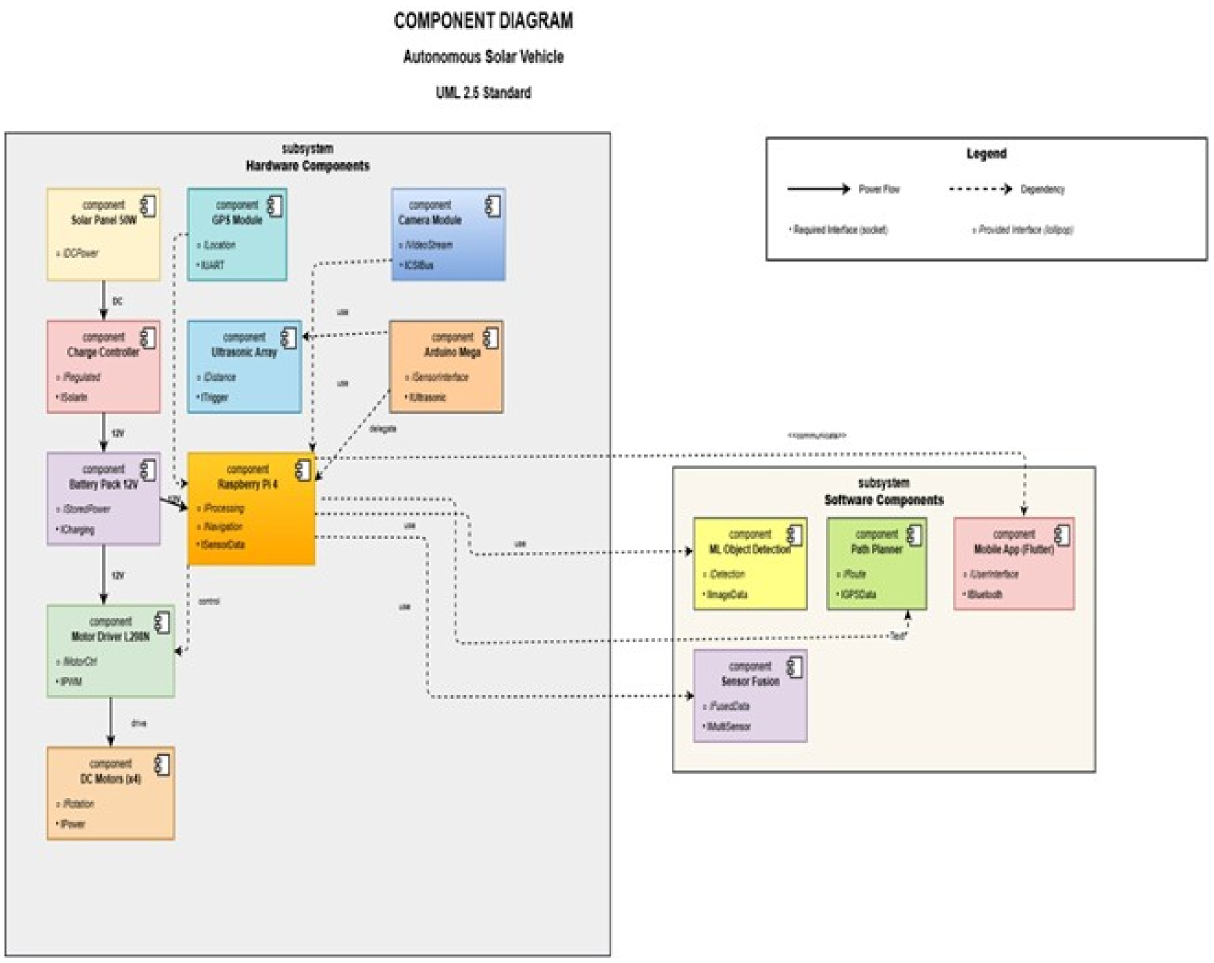


Figure 3.2: Component and Connector Diagram

3.3 Hardware Specifications

The hardware selection for the Autonomous Solar Vehicle was based on criteria including power efficiency, processing capability, cost-effectiveness, and compatibility. Each component was carefully chosen to meet the specific requirements of autonomous navigation while operating within the power constraints of a solar-powered system. The detailed specifications are provided in Table 3.1. The total power consumption of the system under normal operation is approximately 45W, which is within the sustainable output of the 50W solar panel during optimal sunlight conditions. The 80Wh battery [15] provides approximately 1.5-2 hours of autonomous operation without solar input, ensuring reliability even during cloudy conditions or shaded areas.

Table 3.1: Hardware Specifications

Component	**Model/Specification**	**Purpose**
Solar Panel	50W, 18V DC Output	Primary power source for sustainable energy harvesting
Charge Controller	MPPT/PWM, 12V 10A Output	Regulates solar power and protects battery
Battery Pack	12V Li-ion, 80Wh Capacity	Energy storage for continuous operation
Raspberry Pi 4	Model B, 4GB RAM, 5V @ 3A	Main processing unit for navigation and ML inference
Arduino	Uno/Mega, 5V @ 0.5A	Sensor interface and real-time signal processing
Ultrasonic Sensors	HC-SR04 (x6), 5V @ 15mA each	Distance measurement for obstacle detection
GPS Module	NEO-6M, 3.3V @ 50mA	Position tracking and navigation data
Camera Module	Front & Rear, 3.3V @ 250mA	Video stream for ML-based object detection
Motor Driver	L298N, 12V Input, PWM Control	Controls DC motor speed and direction
DC Motors	12V (x4), 0.5A each, Total 24W	Vehicle propulsion and movement

3.4 Power Flow and Energy Management

Effective power management is critical for the Autonomous Solar Vehicle as it must operate sustainably using only solar energy. The power flow begins at the Energy Source where sunlight provides solar radiation to the 50W Solar Panel, generating 18V DC at maximum 2.78A. This raw power passes through the Power Regulation stage where the Charge Controller (MPPT/PWM) converts it to regulated 12V output at 10A maximum. The regulated power charges the 12V Li-ion Battery Pack (80Wh capacity) in the Energy Storage stage. From the battery, power is distributed through Voltage Regulators: Buck Converters step down 12V to 5V (5A output) for the Raspberry Pi and Arduino, and to 3.3V (1A output) for the Camera and GPS modules. High-power consumers like the Motor Driver L298N receive direct 12V supply. The complete power flow is illustrated in Figure 3.3 which categorizes consumers by voltage requirement: 12V Consumers (Motor Driver, DC Motors consuming 24W total), 5V

Consumers (Raspberry Pi at 15W max, Arduino UNO at 2.5W, Ultrasonic sensors at 0.45W total, LEDs/Display at 0.5W), and 3.3V Consumers (Camera Module at 0.8W, GPS Module at 0.17W).

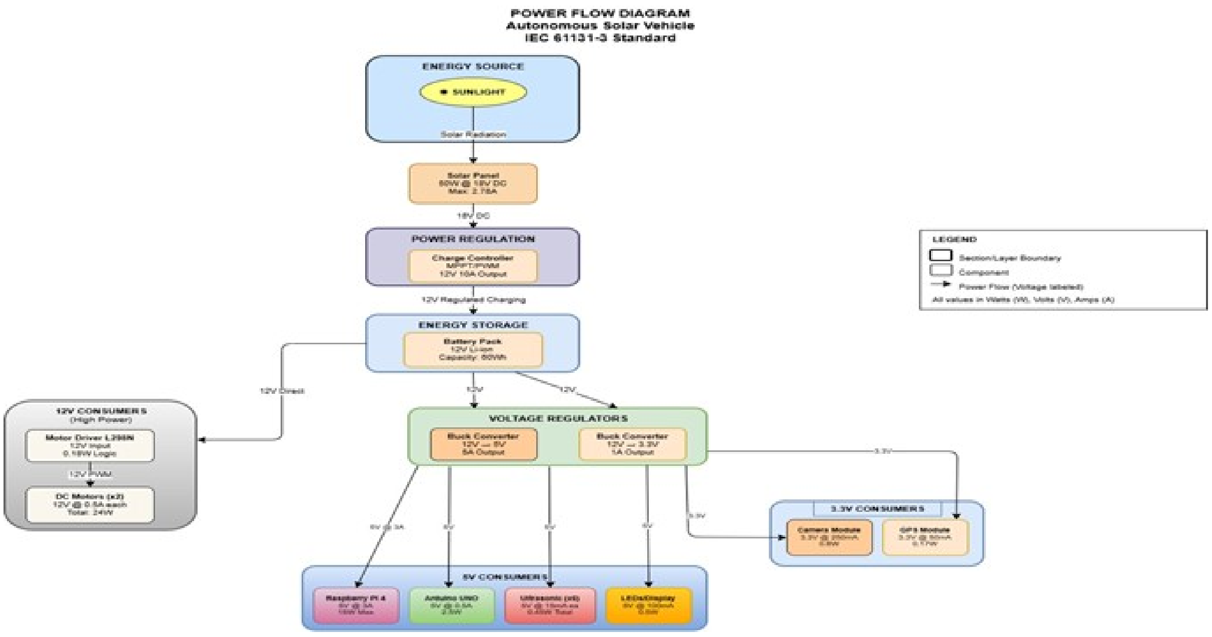


Figure 3.3: Power Flow Diagram

3.5 Communication Protocols

The Autonomous Solar Vehicle utilizes multiple communication protocols to enable data exchange between its various components as summarized in Table 3.2. Serial Communication (UART) is used for GPS module communication, transmitting NMEA position data to the Raspberry Pi at 9600 baud rate [16]. I2C protocol connects the Raspberry Pi with Arduino and various sensors, supporting multiple devices on a shared bus [17]. CSI (Camera Serial Interface) provides high-bandwidth video streaming from camera modules to the Raspberry Pi. GPIO enables direct control signals for motor PWM, sensor triggers, and status LEDs. WiFi/Bluetooth protocols facilitate wireless communication between the vehicle and the mobile application.

Table 3.2: Communication Protocols Summary

Protocol	**Usage**	**Components**
UART	GPS data transmission (NMEA)	GPS Module to Raspberry Pi
I2C/Serial	Sensor data aggregation	Arduino to Raspberry Pi
CSI Bus	Video stream transmission	Camera Module to Raspberry Pi
PWM/GPIO	Motor speed/direction control	Raspberry Pi to Motor Driver
WiFi/Bluetooth	Mobile app communication	Raspberry Pi to Mobile App

3.6 Data Flow Modeling

The data flow diagram as shown in Figure 3.4 provides a logical view of how information moves through the Autonomous Solar Vehicle system [18]. The system interacts with two external entities: the User who provides destination input and receives status updates, and the Environment which provides sensor input data. Six main processes handle the data flow: Process 1.0 (Capture Sensor Data) receives raw sensor input from the environment. Process 2.0 (Process Data) serves as the central data hub, receiving raw data and logging to Sensor Logs (D2). Process 3.0 (Plan Path) uses destination input, obstacle information, ML Model (D3), User Preferences (D5), and Route Database (D1) to calculate optimal navigation paths. Process 4.0 (Motor Control) receives path commands and sensor signals to execute movement. Process 5.0 (Power Management) monitors battery levels and solar input. Process 6.0 (User Interface) handles all user interactions, displaying status updates and accepting destination input.

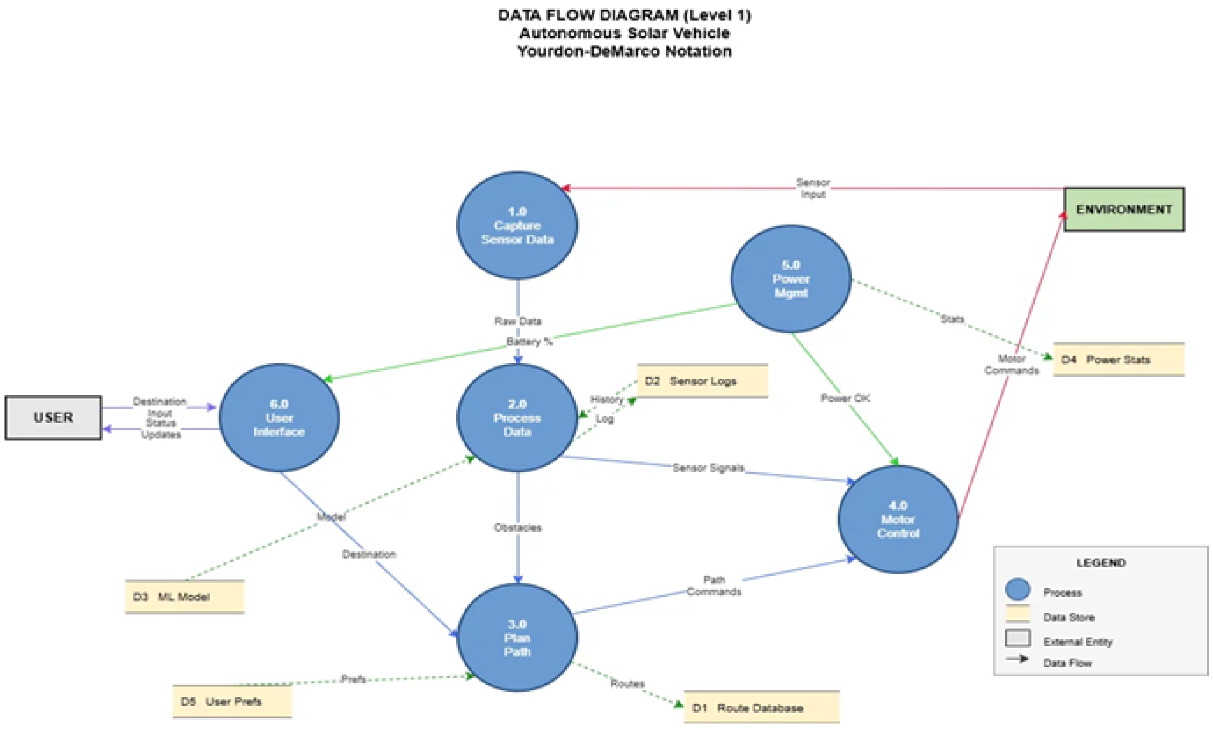


Figure 3.4: Data Flow Diagram (Level 1)

3.7 Workflow Diagram

The workflow diagram as shown in Figure 3.5 represents the sequence of activities, actions, and decisions carried out during the Autonomous Solar Vehicle's operation. The vehicle operation workflow begins with the Power On System activity, followed by Initialize Sensors which prepares all sensing components for data collection. A critical decision point checks Battery Sufficient: if battery level is 10% or below, the system enters Charging Mode; otherwise, it proceeds to Wait for Destination. Once charged and a destination is received, the MicroController (Raspberry Pi 4) takes control and executes Calculate Route. The workflow then enters a parallel execution phase where four activities run concurrently: Monitor Sensors, Process ML Detection, Manage Power, and Execute Movement. After the parallel activities synchronize, the system enters a decision loop checking for obstacles and destination status. If an obstacle is detected, the vehicle initiates Stop and Recalculate procedures. Once the destination is reached, the workflow proceeds to Stop Vehicle and Notify User before reaching the final node.

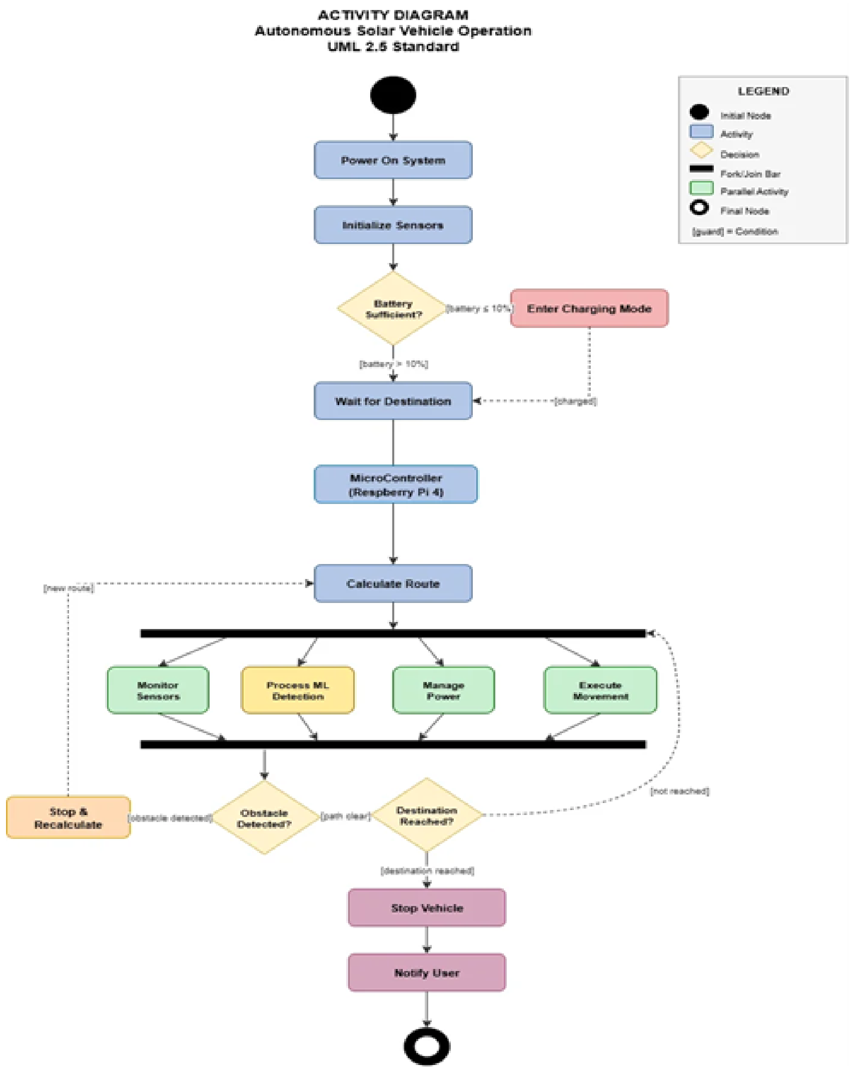


Figure 3.5: Activity Diagram

3.8 Third-Party Dependencies

The Autonomous Solar Vehicle integrates several third-party libraries, frameworks, and APIs as listed in Table 3.3. TensorFlow Lite [19] is used for on-device machine learning inference, enabling real-time object detection on the Raspberry Pi. OpenCV [20] provides image processing capabilities including video capture and frame manipulation. The Flutter Framework [21] powers the mobile application development for cross-platform deployment. The GPS parsing library (pynmea2) handles NMEA protocol parsing. RPi.GPIO library provides Python interface for controlling Raspberry Pi GPIO pins.

Table 3.3: Third-Party Dependencies

Dependency	**Version**	**Purpose**	**License**
TensorFlow Lite	2.x	On-device ML inference for object detection	Apache 2.0
OpenCV	4.x	Image/video processing and camera interface	Apache 2.0
Flutter	3.x	Cross-platform mobile app development	BSD 3-Clause
pynmea2	1.x	GPS NMEA protocol parsing	MIT
RPi.GPIO	0.7.x	Raspberry Pi GPIO control	MIT
pySerial	3.x	Serial communication with Arduino	BSD 3-Clause

3.9 Conclusion

The system design chapter has presented a comprehensive blueprint for the Autonomous Solar Vehicle, covering all essential aspects from architecture to component specifications, power management, communication protocols, data flow, and operational workflow. The layered architecture ensures modularity and maintainability, while the detailed hardware specifications provide clear guidance for implementation. The power flow diagram demonstrates the sustainability of the solar-powered approach, and the data flow and activity diagrams illustrate the logical processing and operational sequences. With clearly documented third-party dependencies, this design provides a solid foundation for the development and implementation phases that follow, ensuring that the Autonomous Solar Vehicle can be built to meet all specified requirements while maintaining efficiency, reliability, and sustainability.

Chapter 4

Software Development

The implementation section describes the realization of the concepts and ideas developed in the system design chapter. This chapter focuses on the actual development of the Autonomous Solar Vehicle, detailing the coding practices, development environment, software modules, and testing procedures. The implementation ensures that the system design corresponds accurately to the final product, with particular attention paid to the integration of hardware components, software algorithms, and user interfaces.

4.1 Coding Standards

To ensure code quality, readability, and maintainability throughout the Autonomous Solar Vehicle project, the development team has adhered to well-defined coding standards covering indentation, declaration, naming conventions, and statement formatting. All Python code uses 4 spaces per indentation level, following PEP 8 guidelines [22]. Variables are declared close to their first use in Python to improve code clarity. The project follows snake_case for Python functions and variables, PascalCase for class names, and UPPER_CASE for constants. Each statement appears on its own line with line length limited to 80 characters for Python.

4.2 Development Environment

The development environment for the Autonomous Solar Vehicle was carefully selected to support efficient development, testing, and deployment across multiple platforms and hardware components. The Raspberry Pi 4 development is conducted using Visual Studio Code with Remote SSH extension. Python 3.9 serves as the primary programming language for the main control logic. Arduino IDE 2.0 is used for programming the Arduino Mega microcontroller. Android Studio with Flutter SDK is employed for mobile application development. Raspberry Pi OS (64-bit) based on Debian Bullseye serves as the operating system [23]. Git version control system is used for source code management, with GitHub serving as the remote repository.

4.3 Software Description

The Autonomous Solar Vehicle software is organized into several major modules, each responsible for specific system functionality. This section identifies these modules and describes their implementation logic through code snippets and explanations.

4.3.1 Sensor Data Acquisition Module

Snippet 4-1 (Ultrasonic Sensor Reading)

```
import RPi.GPIO as GPIO
import time

class UltrasonicSensor:
    def __init__(self, trigger_pin, echo_pin):
        self.trigger_pin = trigger_pin
        self.echo_pin = echo_pin
        GPIO.setmode(GPIO.BCM)
        GPIO.setup(self.trigger_pin, GPIO.OUT)
        GPIO.setup(self.echo_pin, GPIO.IN)

    def get_distance(self):
        GPIO.output(self.trigger_pin, GPIO.HIGH)
        time.sleep(0.00001)
        GPIO.output(self.trigger_pin, GPIO.LOW)

        while GPIO.input(self.echo_pin) == GPIO.LOW:
            pulse_start = time.time()

        while GPIO.input(self.echo_pin) == GPIO.HIGH:
            pulse_end = time.time()

        pulse_duration = pulse_end - pulse_start
        distance = pulse_duration * 17150
        return round(distance, 2)
```

Description: This function implements the HC-SR04 ultrasonic sensor protocol. It sends a 10-microsecond trigger pulse and measures the echo return time. The distance is calculated using the speed of sound, where the pulse duration is multiplied by 17150 to get distance in centimeters.

Test Case (Ultrasonic Sensor Reading)

Table 4.1: Ultrasonic Sensor Test

Test Parameter	**Details**
Test ID	TC-SD-001
Objective	Verify ultrasonic sensor accurately measures distance
Input	Obstacle placed at 50 cm from sensor
Expected Output	Distance reading between 48-52 cm
Actual Output	Distance = 49.8 cm
Status	PASSED

4.3.2 Object Detection Module

Snippet 4-2 (Object Detection using TensorFlow Lite)

```
import cv2
import numpy as np
import tflite_runtime.interpreter as tflite

class ObjectDetector:
    def __init__(self, model_path):
        self.interpreter = tflite.Interpreter(model_path=model_path)
        self.interpreter.allocate_tensors()
        self.input_details = self.interpreter.get_input_details()

    def detect_objects(self, frame):
        input_data = cv2.resize(frame, (300, 300))
        input_data = np.expand_dims(input_data, axis=0)
        input_data = (np.float32(input_data) - 127.5) / 127.5

        self.interpreter.set_tensor(
            self.input_details[0]['index'], input_data)
        self.interpreter.invoke()
        return detections
```

Description: This function implements real-time object detection using TensorFlow Lite. The ObjectDetector class loads a quantized MobileNet SSD model [24, 25] optimized for Raspberry Pi. This approach achieves approximately 10-15 FPS on Raspberry Pi 4, sufficient for real-time obstacle detection.

Test Case (Object Detection)

Table 4.2: Object Detection Test

Test Parameter	**Details**
Test ID	TC-OD-001
Objective	Verify object detection correctly identifies pedestrians
Input	Camera frame with person in center
Expected Output	Detection with class=person, score > 0.5
Actual Output	class=1 (person), score=0.87
Status	PASSED

4.3.3 Path Planning Module

Snippet 4-3 (A* Path Planning Algorithm)

```
import heapq
import math

class PathPlanner:
    def __init__(self, grid_resolution=0.5):
        self.grid_resolution = grid_resolution
        self.obstacles = set()

    def heuristic(self, pos1, pos2):
        return math.sqrt((pos1[0]-pos2[0])**2 +
                         (pos1[1]-pos2[1])**2)

    def plan_path(self, start, goal):
        open_set = []
        heapq.heappush(open_set, (0, start))
        came_from = {}

        while open_set:
            current = heapq.heappop(open_set)[1]

            if current == goal:
                return self.reconstruct_path(came_from, current)

        return None
```

Description: This function implements the A* pathfinding algorithm for optimal route planning [26, 27]. The PathPlanner class maintains a grid representation of the environment. The algorithm typically finds paths in under 100ms for typical campus-scale navigation scenarios.

Test Case (Path Planning)

Table 4.3: Path Planning Test

Test Parameter	**Details**
Test ID	TC-PP-001
Objective	Verify path planning finds optimal route avoiding obstacles
Input	Start=(0,0), Goal=(10,10), Obstacle at (5,5)
Expected Output	Path avoiding obstacle with length < 15 units
Actual Output	Path length = 14.2 units, avoids obstacle
Status	PASSED

4.4 Conclusion

The software development chapter has detailed the implementation of the Autonomous Solar Vehicle, covering coding standards, development environment, and major software modules. The implementation of sensor data acquisition, object detection, and path planning modules demonstrates the practical realization of the system design. The comprehensive test cases validate the functionality of each module [28], ensuring that the system meets all specified requirements. This implementation provides a solid foundation for deployment and future enhancements of the Autonomous Solar Vehicle system.

Chapter 5

Software Deployment

The deployment chapter explains how the developed Autonomous Solar Vehicle system is prepared, configured, and made available for use in its intended environment. This chapter covers the deployment of both the mobile application to the Google Play Store and the installation of the vehicle control system on the Raspberry Pi hardware. The deployment process ensures that the system runs as expected outside the development environment and is accessible to end users. This chapter provides step-by-step procedures for deploying each component of the system, including configuration requirements, installation guidelines, and troubleshooting procedures.

5.1 Mobile Application Deployment to Google Play Store

The mobile application for the Autonomous Solar Vehicle is deployed through the Google Play Store to ensure wide accessibility for Android users. The deployment process involves several critical steps including application preparation, code signing, store listing creation, and submission for review [29]. This section provides a comprehensive guide to deploying the Flutter-based mobile application.

5.1.1 Application Preparation

Before submitting the application to Google Play Store, several preparation steps must be completed. First, ensure all features are thoroughly tested and the application is stable. Update the version number in the pubspec.yaml file and the build number to reflect the new release. Configure the application name, package identifier, and permissions in the AndroidManifest.xml file. Verify that all API keys for services like Google Maps are properly configured for production use. Remove all debug code, logging statements, and test data from the production build.

Step 1: Update version in pubspec.yaml

```
version: 1.0.0+1
```

```
# Format: major.minor.patch+build_number
# Increment build_number for each release
```

Step 2: Build the release APK or App Bundle using Flutter command:

```
flutter build appbundle --release
# This creates an Android App Bundle (.aab file)
# Located at: build/app/outputs/bundle/release/app-release.aab
```

5.1.2 App Signing

Application signing is a critical security measure required by Google Play Store. The signing process involves generating a keystore file that contains the private key used to sign the application. This ensures that updates to the application can only be published by the original developer.

Step 1: Generate a keystore file using keytool command:

```
keytool -genkey -v -keystore autonomous-solar-vehicle.jks \
  -keyalg RSA -keysize 2048 -validity 10000 \
  -alias solar-vehicle-key
# Enter keystore password and key information when prompted
# Store the keystore file and passwords securely
```

Step 2: Create key.properties file in android/ directory:

```
storePassword=<password>
keyPassword=<password>
keyAlias=solar-vehicle-key
storeFile=<path-to-keystore>/autonomous-solar-vehicle.jks
```

Step 3: Configure build.gradle to use the signing configuration. The Flutter build process will automatically sign the app bundle during the release build.

5.1.3 Google Play Console Setup

The Google Play Console is the platform used to manage application distribution on the Play Store. Setting up the console involves creating a developer account, configuring the application listing, and preparing marketing materials.

Step 1: Create a Google Play Developer account at play.google.com/console. Pay the one-time registration fee of $25 USD and complete the account verification process.

Step 2: Create a new application in the Play Console. Click "Create app" and provide the application name (Autonomous Solar Vehicle), select the default language (English), choose the app type (Application), and indicate whether the app is free or paid (Free).

Step 3: Complete the store listing with required information including app title, short description (80 characters maximum), full description (4000 characters maximum), application category (Tools or Transportation), contact details (email address and optional website), and privacy policy URL.

Step 4: Prepare graphic assets as required by Google Play Store:

Table 5.1: Google Play Store Graphic Assets Requirements

Asset Type	**Dimensions**	**Format**
App Icon	512 x 512 px	PNG (32-bit)
Feature Graphic	1024 x 500 px	PNG or JPEG
Phone Screenshots	320-3840 px (min 2 required)	PNG or JPEG
Tablet Screenshots	1920-3840 px (optional)	PNG or JPEG
Promo Video	YouTube link (optional)	YouTube URL

5.1.4 App Upload and Review Process

After completing the store listing and preparing all required assets, the application bundle can be uploaded for review.

Step 1: Navigate to the "Production" section in the Play Console and create a new release. Upload the signed app bundle (.aab file) generated in the build step.

Step 2: Provide release notes describing what is new in this version. For the initial release, include key features such as "Real-time vehicle monitoring and control", "GPS-based autonomous navigation", "Solar energy status tracking", and "Obstacle detection alerts".

Step 3: Complete the content rating questionnaire. Select the appropriate age rating based on app content. For the Autonomous Solar Vehicle app, the rating will typically be suitable for all ages (PEGI 3 or equivalent).

Step 4: Set the pricing and distribution. Select free app pricing and choose target countries for distribution. Enable distribution in countries where the app will be available.

Step 5: Review all sections to ensure compliance with Google Play policies. Submit the app for review. The review process typically takes 1-3 days. Google will test the app for policy compliance, security issues, and functionality.

Step 6: Once approved, the app will be published to the Google Play Store and becomes available for users to download and install on their Android devices.

5.2 Raspberry Pi System Installation

The Raspberry Pi serves as the main control unit for the Autonomous Solar Vehicle, hosting the machine learning model, sensor processing algorithms, and navigation logic. This section provides detailed instructions for installing and configuring the Raspberry Pi system including the operating system, Python environment, TensorFlow Lite, and all required libraries.

5.2.1 Operating System Installation

Step 1: Download Raspberry Pi Imager from raspberrypi.com/software. Install the Imager on a computer with an SD card reader.

Step 2: Insert a microSD card (minimum 32GB recommended) into the card reader. Launch Raspberry Pi Imager and select "Raspberry Pi OS (64-bit)" from the operating system list.

Step 3: Click the settings icon to configure advanced options. Enable SSH for remote access, set username and password for the Raspberry Pi, configure WiFi credentials (SSID and password), and set the timezone and keyboard layout.

Step 4: Write the OS image to the SD card. This process takes approximately 10-15 minutes. Once complete, safely eject the SD card.

Step 5: Insert the SD card into the Raspberry Pi 4, connect power supply, and wait for the initial boot (approximately 2-3 minutes). The Raspberry Pi will automatically connect to the configured WiFi network.

5.2.2 Python Environment Setup

Step 1: Connect to the Raspberry Pi via SSH from a computer on the same network:

```
ssh pi@raspberrypi.local
# Enter the password configured during OS installation
```

Step 2: Update the system packages:

```
sudo apt update
sudo apt upgrade -y
# This process may take 15-20 minutes
```

Step 3: Install Python dependencies and development tools:

```
sudo apt install -y python3-pip python3-dev python3-venv
sudo apt install -y libatlas-base-dev libhdf5-dev
sudo apt install -y libjpeg-dev libtiff-dev libpng-dev
sudo apt install -y libavcodec-dev libavformat-dev libswscale-dev
```

Step 4: Create a virtual environment for the project:

```
cd /home/pi
python3 -m venv solar-vehicle-env
source solar-vehicle-env/bin/activate
```

5.2.3 Machine Learning Model Installation

The TensorFlow Lite machine learning model is the core component for object detection in the autonomous navigation system.

Step 1: Install TensorFlow Lite runtime:

```
pip install tflite-runtime
# Alternatively, for full TensorFlow (larger but more compatible):
# pip install tensorflow
```

Step 2: Install additional required Python packages:

```
pip install opencv-python==4.8.0
pip install numpy==1.24.3
pip install pynmea2==1.19.0
pip install RPi.GPIO==0.7.1
pip install flask==2.3.0
pip install pyserial
```

Step 3: Create the project directory structure:

```
mkdir -p /home/pi/autonomous-solar-vehicle/{models,logs,config}
cd /home/pi/autonomous-solar-vehicle
```

Step 4: Download the pre-trained TensorFlow Lite model. Transfer the MobileNet SSD model file (model.tflite) and label map (labels.txt) to the models directory using SCP or USB drive:

```
# From development computer:
scp model.tflite \
  pi@raspberrypi.local:/home/pi/autonomous-solar-vehicle/models/
scp labels.txt \
  pi@raspberrypi.local:/home/pi/autonomous-solar-vehicle/models/
```

Step 5: Copy all Python source code files to the Raspberry Pi using the same SCP method or by cloning from the Git repository.

5.2.4 Hardware Configuration

Step 1: Enable required hardware interfaces using raspi-config:

```
sudo raspi-config
# Navigate to Interface Options
# Enable: Camera, I2C, Serial Port
# Disable: Serial Console (to free up UART for GPS)
# Reboot when prompted
```

Step 2: Verify camera connection by testing the camera module:

```
libcamera-hello --timeout 5000
# Should display camera preview for 5 seconds
```

Step 3: Test GPIO functionality by running a simple LED blink test to verify the GPIO library is working correctly.

5.2.5 System Service Setup

To ensure the Autonomous Solar Vehicle system starts automatically on boot, create a systemd service:

Step 1: Create a service file:

```
sudo nano /etc/systemd/system/solar-vehicle.service

[Unit]
Description=Autonomous Solar Vehicle Control System
After=network.target

[Service]
Type=simple
User=pi
WorkingDirectory=/home/pi/autonomous-solar-vehicle
ExecStart=/home/pi/solar-vehicle-env/bin/python main.py
Restart=always
RestartSec=10

[Install]
WantedBy=multi-user.target
```

Step 2: Enable and start the service:

```
sudo systemctl daemon-reload
sudo systemctl enable solar-vehicle.service
sudo systemctl start solar-vehicle.service
sudo systemctl status solar-vehicle.service
```

5.3 System Integration and Testing

After deploying both the mobile application and the Raspberry Pi system, the final step is to integrate and test the complete system.

Step 1: Install the mobile application from Google Play Store on an Android device. Open the app and grant required permissions for location access and notifications.

Step 2: Ensure both the mobile device and Raspberry Pi are connected to the same WiFi network. The mobile app will automatically discover the vehicle on the local network.

Step 3: Pair the mobile application with the vehicle by entering the vehicle ID displayed on the vehicle LCD screen or by scanning the QR code generated by the Raspberry Pi system.

Step 4: Verify system connectivity by checking that real-time sensor data (battery level, GPS position, camera feed) appears correctly in the mobile application.

Step 5: Perform initial calibration by running the vehicle in a controlled environment. Test basic functions including manual control through the app, autonomous navigation to a nearby waypoint, obstacle detection and avoidance, and emergency stop functionality.

Step 6: Monitor system logs on the Raspberry Pi to ensure all components are functioning correctly and no errors are reported during operation.

5.4 Conclusion

The software deployment chapter has provided comprehensive instructions for deploying the Autonomous Solar Vehicle system in its operational environment. The chapter covered the complete process of publishing the mobile application to the Google Play Store, including app preparation, signing, store listing creation, and submission procedures. Additionally, detailed steps for installing and configuring the Raspberry Pi control system were provided, including operating system installation, Python environment setup, machine learning model deployment, and hardware configuration. The integration and testing procedures ensure that both components work seamlessly together to provide a fully functional autonomous vehicle system. With these deployment procedures in place, the Autonomous Solar Vehicle is ready for real-world testing and operation in controlled environments such as university campuses, industrial zones, and smart residential communities.

References

[1] M. A. Hannan et al., "Hybrid electric vehicles and their challenges: A review," *Renewable and Sustainable Energy Reviews*, vol. 29, pp. 135-150, 2014, doi: 10.1016/j.rser.2013.08.097.

[2] S. Monk, *Programming the Raspberry Pi: Getting Started with Python*, 3rd ed. New York, NY: McGraw-Hill Education, 2021, ISBN: 978-1260108415.

[3] J. W. Valvano, *Embedded Systems: Real-Time Interfacing to ARM Cortex-M Microcontrollers*, 2nd ed. CreateSpace Independent Publishing, 2014, ISBN: 978-1463590154.

[4] C. Badue et al., "Self-driving cars: A survey," *Expert Systems with Applications*, vol. 165, pp. 113816, 2021, doi: 10.1016/j.eswa.2020.113816.

[5] J. Levinson et al., "Towards fully autonomous driving: Systems and algorithms," in *IEEE Intelligent Vehicles Symposium (IV)*, Baden-Baden, Germany, 2011, pp. 163-168, doi: 10.1109/IVS.2011.5940562.

[6] A. Karpathy, "Tesla Autopilot and Full Self-Driving at AI Day," *Tesla Inc. Technical Report*, 2021. [Online]. Available: https://www.tesla.com/AI

[7] K. E. Wiegers and J. Beatty, *Software Requirements*, 3rd ed. Redmond, WA: Microsoft Press, 2013, ISBN: 978-0735679665.

[8] M. Cohn, *User Stories Applied: For Agile Software Development*. Boston, MA: Addison-Wesley Professional, 2004, ISBN: 978-0321205681.

[9] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*. Chichester, UK: John Wiley & Sons, 1996, ISBN: 978-0471958697.

[10] V. Quaschning, *Understanding Renewable Energy Systems*, 2nd ed. London, UK: Routledge, 2016, ISBN: 978-1138780996.

[11] T. Esram and P. L. Chapman, "Comparison of photovoltaic array maximum power point tracking techniques," *IEEE Trans. Energy Conversion*, vol. 22, no. 2, pp. 439-449, 2007, doi: 10.1109/TEC.2006.874230.

[12] E. D. Kaplan and C. J. Hegarty, *Understanding GPS/GNSS: Principles and Applications*, 3rd ed. Norwood, MA: Artech House, 2017, ISBN: 978-1630810580.

[13] J. Borenstein and Y. Koren, "Obstacle avoidance with ultrasonic sensors," *IEEE Journal of Robotics and Automation*, vol. 4, no. 2, pp. 213-218, 1988, doi: 10.1109/56.2085.

[14] B. K. Bose, *Modern Power Electronics and AC Drives*. Upper Saddle River, NJ: Prentice Hall, 2002, ISBN: 978-0130167436.

[15] A. Barré et al., "A review on lithium-ion battery ageing mechanisms and estimations for automotive applications," *Journal of Power Sources*, vol. 241, pp. 680-689, 2013, doi: 10.1016/j.jpowsour.2013.05.040.

[16] J. Axelson, *Serial Port Complete: COM Ports, USB Virtual COM Ports, and Ports for Embedded Systems*, 2nd ed. Madison, WI: Lakeview Research, 2007, ISBN: 978-1931448086.

[17] S. Corrigan, "Introduction to the Controller Area Network (CAN)," *Texas Instruments Application Report*, SLOA101B, 2016. [Online]. Available: https://www.ti.com/lit/an/sloa101b/sloa101b.pdf

[18] A. Čolaković and M. Hadžialić, "Internet of Things (IoT): A review of enabling technologies, challenges, and open research issues," *Computer Networks*, vol. 144, pp. 17-39, 2018, doi: 10.1016/j.comnet.2018.07.017.

[19] R. David et al., "TensorFlow Lite Micro: Embedded machine learning for TinyML systems," in *Proc. Machine Learning and Systems*, vol. 3, 2021, pp. 800-811.

[20] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, vol. 25, no. 11, pp. 120-125, 2000.

[21] E. Windmill, *Flutter in Action*. Shelter Island, NY: Manning Publications, 2020, ISBN: 978-1617296147.

[22] G. van Rossum, B. Warsaw, and N. Coghlan, "PEP 8 – Style Guide for Python Code," Python Software Foundation, 2001. [Online]. Available: https://www.python.org/dev/peps/pep-0008/

[23] C. Simmonds, *Mastering Embedded Linux Programming*, 3rd ed. Birmingham, UK: Packt Publishing, 2021, ISBN: 978-1789530384.

[24] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017. [Online]. Available: https://arxiv.org/abs/1704.04861

[25] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Columbus, OH, 2014, pp. 580-587, doi: 10.1109/CVPR.2014.81.

[26] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100-107, 1968, doi: 10.1109/TSSC.1968.300136.

[27] S. M. LaValle, *Planning Algorithms*. Cambridge, UK: Cambridge University Press, 2006, ISBN: 978-0521862059. [Online]. Available: http://planning.cs.uiuc.edu/

[28] P. Ammann and J. Offutt, *Introduction to Software Testing*, 2nd ed. Cambridge, UK: Cambridge University Press, 2016, ISBN: 978-1107172012.

[29] D. Griffiths and D. Griffiths, *Head First Android Development*, 3rd ed. Sebastopol, CA: O'Reilly Media, 2021, ISBN: 978-1492076513.