RoboNav

SE4450: Software Engineering Design Project Proposal

Faculty Advisor: **Dr. Yili (Kelly) Tang**

Email: ytang564@uwo.ca

Team 7:

Bryson Crook
bcrook4@uwo.ca
251217987

Christopher Higgins
chiggi24@uwo.ca
251245390

Mohamed El Dogdog
meldogdo@uwo.ca
251239204

Seth Langendoen
slangend@uwo.ca
251226610

October 9, 2024

GitHub Repo: https://github.com/meldogdo/RoboNav

Revision History

T – Team | C – Chris | M – Mohamed | B – Bryson | S - Seth

Date	Version	Description	Authors
October 1, 2024	0.1	Formatted Word Doc	T
October 3, 2024	0.1	Logo Completed	B
October 4, 2024	0.1	Advisor contacted and agreed	M
October 5, 2024	0.1	Background completed	B, C
October 6, 2024	0.1	Introduction completed	B, C
October 6, 2024	0.1	Goals completed	B, C
October 6, 2024	0.1	Constraints completed	B, C
October 7, 2024	0.1	Methodologies/ Processes completed	B, C, M
October 8, 2024	0.1	Deliverables and Resources	B, C, M
October 8, 2024	0.1	GANT Chart completed	B, C
October 8, 2024	0.1	Completed Responsibility Matrix	B, C
October 9, 2024	0.1	Completed Justification	M, C
October 9, 2024	0.1	Glossary completed	B, C, M
October 9, 2024	0.1	References Completed	B, C, M
October 9, 2024	0.1	Fixed formatting errors	B, C, M
October 9, 2024	0.1	Revised the whole document	B, C, M

Table of Contents

1. Introduction (Problem Statement)

In the rapidly advancing field of robotics, communication and control systems are critical for optimizing performance and ensuring the smooth operation of autonomous systems. As industries continue to embrace automation, there is an increasing demand for flexible, reliable backend APIs that enable seamless interaction between robots and control systems. This project aims to address the limitations of the current backend API used by our university's autonomous robots, enhancing its functionality to create a more efficient, scalable, and dependable framework.

Currently in place is a web-based remote-control system that allows users to manage robotic movements and view limited status information. However, this system faces communication delays, limited data insights, and difficulty scaling with multiple robots. Our main objective is to significantly improve the backend API by introducing advanced features such as real-time data processing, enhanced robot navigation, and robust cloud infrastructure. These enhancements will reduce latency, improve communication reliability, and provide real-time updates on the robots' location, operational status, and battery levels.

The upgraded API will allow more interactions with robots, including precise navigation between designated points on an indoor map, handling real-time robot status updates, and ensuring seamless integration with cloud services for data storage and retrieval. Additionally, integrated collision detection and obstacle avoidance prevent operational hazards during robot movements. With these improvements, users will be able to monitor and control multiple robots efficiently, with improved safety and operational accuracy.

Alongside backend improvements, we aim to develop a mobile application providing an intuitive UI for controlling and monitoring the robots. The app will offer real-time robot tracking, status and battery visualization, ensuring users can manage robots effectively. This user-friendly interface will reduce the learning curve for new users and make it easier to access and control robotic systems remotely.

By focusing this project on both backend API enhancements and the development of a mobile app, this project will deliver a comprehensive solution that ensures scalable, flexible, and reliable robot control. It will address the current limitations and provide a foundation for future development.

2. Background

2.1. Robotics and Control Systems

The field of robotics has seen significant growth in automation, driven by advancements in control systems, machine learning, and cloud computing. Effective communication between robots and control software is critical for operational efficiency, especially in engineering laboratories where precision is vital. Robotics in autonomous systems have traditionally relied on web-based interfaces for control, but these systems often suffer from latency, lack of scalability, and limited flexibility in handling real-time data. Our project seeks to address these shortcomings by enhancing backend APIs for better performance and reliability.

2.2. Previous Attempts

Current control systems for autonomous robots often employ standard web APIs that allow remote control of movements and limited feedback from sensors such as location or battery level. While functional, these systems are not optimized for complex navigation tasks or real-time status updates across various platforms. One of the limitations of these systems is their inability to efficiently process and react to real-time data, leading to latency and poor user experience in scenarios that require instant communication, such as obstacle detection or route planning.

Several existing solutions in the market, such as ROS (Robot Operating System), provide a framework for robotic control, but they are typically not optimized for lightweight mobile control applications. ROS systems are designed for larger-scale automation but can be too complex for simplified mobile implementations, which is the focus of our project.

2.3. Emerging Solutions

Recent advancements in cloud robotics offer new opportunities for improving communication efficiency. Cloud-based solutions allow robots to store and retrieve data in real time, reducing the computational load on individual devices. Integrating cloud infrastructure into the existing robotic control system will enable the processing of large amounts of sensor data, improving real-time navigation capabilities and reducing latency.

Additionally, collision detection using AI algorithms is gaining traction. By implementing machine learning models and leveraging tools like TensorFlow and PyTorch, more advanced autonomous navigation systems can be created. These tools enable robots to learn from their environments, predict obstacles, and adjust routes on the fly, creating a safer and more efficient workflow for users.

2.4. Project's Role

Our project will build upon these emerging trends by expanding the capabilities of the backend API currently used for robot control. We aim to create a more robust system that not only facilitates robot navigation between designated points on a map but also retrieves status updates like location, battery life, and real-time collision detection data. By leveraging cloud-based storage, we will enable seamless communication between the robots and the user interface, both on the web and mobile applications.

The mobile app will allow users to send commands to the robots, receive live updates on their location and status, and process critical data through an intuitive and user-friendly interface. This level of real-time interaction, coupled with improved backend processing, will significantly enhance the user experience while ensuring the robots' autonomous functionality.

2.5. Comparative Analysis

When comparing existing systems, our solution stands out due to its focus on mobile integration, cloud-based real-time updates, and enhancements to the backend API for better scalability and performance. Many prior systems have successfully implemented backend APIs for robot control, but they have lacked flexibility, real-time processing, and mobile-friendly designs. Our project aims to bridge this gap by delivering a system optimized for both performance and user accessibility.

3. Objectives and Scope

3.1. Goals

Develop a Robust Backend API

Create a scalable API to enable seamless communication between the mobile app and autonomous robots, supporting functionalities such as retrieving robot status, sending movement commands, and logging data.

Implement Real-Time Data Handling

Ensure real-time data exchange between the mobile application and robots, providing timely updates on location, battery status, and operational status using low-latency communication technologies like WebSocket.

Enhance Robot Navigation Capabilities

Integrate pathfinding and obstacle detection algorithms to improve the robots' autonomous navigation in designated indoor maps, utilizing sensor fusion techniques for accurate data interpretation.

Create a User-Friendly Mobile Application

Develop an intuitive mobile app interface that allows users to easily control and monitor robots, providing real-time updates and visualizing their status on indoor maps.

Establish Cloud Infrastructure

Set up a cloud-based environment for secure data storage, ensuring access to robot status, historical data, and system logs.

Conduct Comprehensive Testing and Validation

Implement a thorough testing plan, including unit testing, integration testing, and user acceptance testing, to ensure system reliability and functionality while collecting feedback.

3.2. Constraints

Time Constraints

The project must be completed within a 6-month timeframe, including development, testing, and the final demo. This limits the scope of features that can be realistically developed and tested.

Resource Limitations

Limited access to specific hardware components or external datasets may restrict the ability to fully implement certain features. The project will focus primarily on software development rather than hardware. Additionally, the project must account for the limited processing power and memory available on the robots, making cloud integration essential for handling heavy computations, such as collision detection and real-time data processing.

Technical Skill Requirements

The project relies on the team's proficiency in specific programming languages (Python for backend and mobile frameworks), which may constrain the choice of technologies used in development.

Budget Constraints

Budget limitations may affect the choice of cloud services or third-party tools, requiring consideration of cost-effective solutions. Depending on the scale of the robots and the amount of real-time data being processed, the cost of cloud storage, computing, and bandwidth may also be a constraint, requiring efficient resource usage.

Real-Time Data Handling

Latency in communication between the robots and the control system must be minimized, imposing constraints on network performance and demanding optimization of both the backend API and cloud infrastructure. Potential integration challenges between the backend API and mobile application, particularly regarding real-time data synchronization, must also be anticipated and managed.

User-Friendly Interface

The mobile app must be simple enough for non-technical users to operate, requiring careful consideration of UI/UX design principles. Complex technical details should be hidden from the user to reduce the learning curve.

Indoor Mapping Limitations

The project's use of indoor maps (e.g., ACEB Atrium) must consider the resolution and accuracy of the map data, ensuring that robots can navigate reliably, even with potential limitations in indoor positioning systems.

Compatibility with Existing Systems

The new backend API must integrate seamlessly with the existing web-based control system and other legacy software that the robots might be using.

Security Concerns

Storing and retrieving data in the cloud introduces security risks, especially when controlling autonomous systems. Robust authentication, data encryption, and secure communication protocols must be implemented.

Testing and Validation Time

The limited timeframe for testing and validation may impact the thoroughness of acceptance testing, potentially affecting the final product quality.

4. Methodologies/Processes

4.1. Project Planning and Requirements

- **Gathering Objectives**: Clearly define the objectives of the robotic control system and the mobile app.
- **Stakeholder Engagement**: Conduct interviews and surveys with stakeholders to gather requirements.
- **Documentation**: Create a Software Requirements Specification (SRS) document detailing functional and non-functional requirements.

4.2. System Architecture Design

- **Architecture Overview**: Design a scalable architecture that integrates the backend API, mobile application, and robotic systems.
- **Technology Stack Selection**: Choose appropriate technologies for the backend (e.g., Node.js, Python Flask) and the mobile app (e.g., React Native, Flutter).
- **Data Flow Diagrams**: Create data flow diagrams to visualize the interaction between components.

4.3. Backend Development

- **API Development**: Develop RESTful APIs for communication between the mobile app and the robotic system.
    - **Endpoints**: Define endpoints for controlling robots, retrieving status, and managing user sessions.
    - **Error Handling**: Implement robust error handling to ensure reliability and fault tolerance.
- **Technical Innovations**:
    - **Real-Time Collision Detection:** Utilize robot location data (e.g., (x, y) coordinates) and telemetry from APIs to provide real-time collision detection, processing this data in the backend for obstacle avoidance and safe navigation.
    - **Path Optimization Comparison:** Develop and test alternative pathfinding algorithms (A*, Dijkstra's, etc.) in a simulated environment, comparing them with existing robot algorithms to identify potential improvements in efficiency and safety.

    - **Data Fusion Technologies**: Combine information from various robot APIs (e.g., location, battery levels) to create a more accurate environmental map, improving navigation and obstacle detection by leveraging data fusion techniques.

- **Database Integration**: Set up a cloud database (e.g., Firebase, AWS RDS) for storing robot status, locations, and battery information.

4.4. Mobile App Development

- **User Interface Design**: Create user-friendly wireframes and design mock-ups to enhance usability.
- **Implementation**: Develop the mobile application with functionalities such as:
    - **Real-Time Updates**: Display real-time location and status of robots.
    - **Control Interface**: Provide intuitive controls for moving robots to designated points on the map.
- **Testing**: Conduct unit and integration testing to ensure app functionality and performance.

4.5. Testing and Validation

- **Testing Methodologies**: Employ various testing methodologies, including:
    - **Unit Testing**: Verify individual components of the backend and mobile app.
    - **Integration Testing**: Ensure that different modules work together seamlessly.
    - **User Acceptance Testing (UAT)**: Engage users in testing the application to gather feedback and ensure it meets their needs.
- **Iterative Improvement**: Implement feedback loops to refine features and address any issues promptly.

4.6. Deployment and Maintenance

- **Deployment Strategy**: Develop a deployment plan for the backend services and mobile application, ensuring minimal disruption.
- **Continuous Integration/Continuous Deployment (CI/CD)**: Implement CI/CD practices to streamline updates and maintain code quality.

4.7. Conclusion

This methodology outlines a systematic approach to developing a robust robotic control system and mobile application. By focusing on clear planning, sound technical practices, and the potential for innovation, this project aims to provide a comprehensive solution that meets user needs while advancing technological capabilities in robotics.

Additionally, the processes align with the software development life cycle (SDLC), ensuring a structured flow from planning and requirements gathering to deployment and maintenance. This adherence to the SDLC enhances accountability and communication among team members, leading to a more reliable final product.

5. Deliverables

5.1. Important Milestones

- Project Proposal (October 9th)
- Walkthrough (November 6th)
- Sprint 1 and 2 Plan Report (November 6th)
- Demo of Release 1 (December 4th)
- Presentation and Final Demo (March 19th)
- Final Report Retrospective (March 26th)

5.2. Deliverables

- Project Estimation Report
- Software Requirements Specification (SRS)
- Software Design Specification (SDS)
- API Documentation
- Mobile App Wireframes and Design Mock-ups
- Mobile App Source Code
- Backend Source Code
- Test Plan and Test Cases
- Cloud Infrastructure Setup
- Mobile-Backend Integration Plan
- Verification and Validation Document
- User Manual (for both API and mobile app)

6. Resources

6.1. Hardware Resources

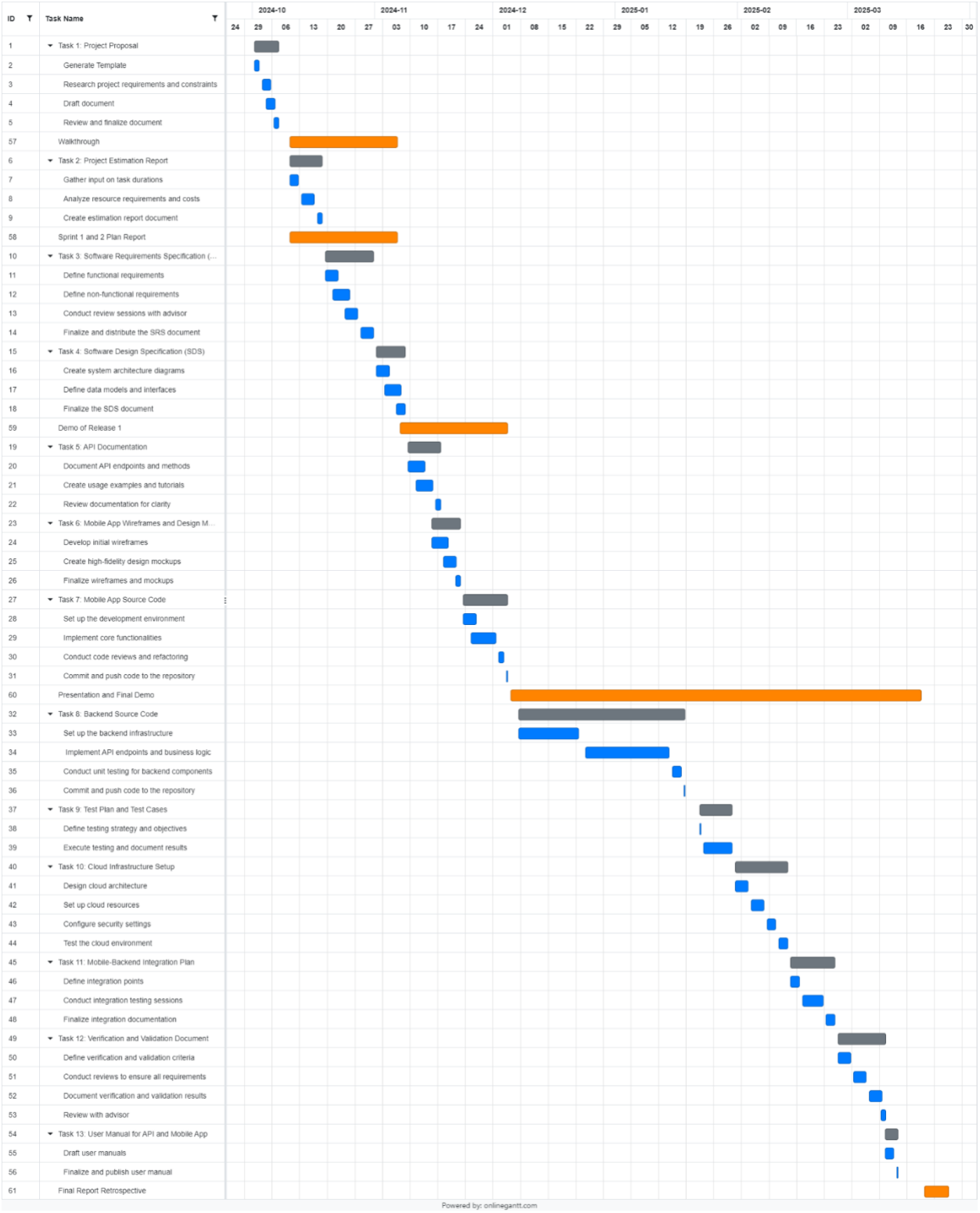
- **Robotic Units**: Autonomous robots equipped with essential sensors for navigation and collision detection will be used for development and testing.
- **Mobile Devices**: Smartphones or tablets (both Android and iOS) will be required to test the mobile app, ensuring cross-platform compatibility and ease of control over the robotic systems.
- **Cloud Infrastructure**: A cloud-based server platform (e.g., AWS, Google Cloud) will be used to host the backend API, manage data storage, and handle real-time communication between the robots and mobile devices.

6.2. Software Resources

- **Development Tools:**
    - **Backend Development:** Technologies like Python (or Node.js) will be used to develop the backend API, manage data, and handle real-time communication with the robots. The cloud will also be used for secure data storage and computations.
    - **Mobile Development Frameworks:** A cross-platform framework like React Native or Flutter will be utilized to build the mobile application, providing seamless control and real-time updates of the robots.
- **Database Solutions:** A cloud-hosted database (e.g., Firebase, AWS RDS) will store robot status, log data, and historical information, ensuring scalability and availability of critical data.
- **Testing Tools:** Various testing methodologies will be employed to validate API functionality, mobile app performance, and overall system integration. This includes unit tests, integration tests, and user acceptance testing to ensure the product meets all functional requirements.
- **Version Control:**
    - **Git and GitHub:** For source code management, allowing for collaboration among team members and maintaining version history.
- **Project Management Tools:**
    - **Trello or Jira:** For task management and tracking project progress, ensuring that all milestones are met within the timeline.

7. Planning

7.1. Schedule (Ghantt Chart)



7.2. Matrix of Responsibilities (RACI)

R – Responsible | A – Accountable | C – Consulted | I – Informed

Tasks/Responsibilities	Bryson Crook	Chris Higgins	Mohamed El Dogdog	Seth Langendoen
Task 1: Project Proposal				
Generate Template	R/A	I	I	I
Research project requirements and constraints	C	C	R/A	C
Draft document	R/A	R	I	I
Review and finalize document	R/A	R/A	R/A	R/A
Task 2: Project Estimation Report				
Gather input on task durations	A	I	R	I
Analyze resource requirements and costs	R	C	A	C
Create estimation report document	R/A	C	I	I
Task 3: Software Requirements Specification (SRS)				
Define functional requirements	R	C	A	C
Define non-functional requirements	R	I	C	A
Conduct review sessions with advisor	I	C	R/A	I
Finalize and distribute the SRS document	R/A	R/A	R/A	R/A
Task 4: Software Design Specification (SDS)				

Create system architecture diagrams	R/A	C	R/A	I
Define data models and interfaces	C	R	I	A
Finalize the SDS document	R/A	R/A	R/A	R/A
Task 5: API Documentation				
Document API endpoints and methods	C	R/A	R/A	I
Create usage examples and tutorials	A	C	R	I
Review documentation for clarity	R/A	R/A	R/A	R/A
Task 6: Mobile App Wireframes and Design Mockups				
Develop initial wireframes	R/A	C	I	R/A
Create high-fidelity design mockups	R/A	I	R/A	C
Finalize wireframes and mockups	R/A	R/A	R/A	R/A
Task 7: Mobile App Source Code				
Set up the development environment	C	R/A	C	R/A
Implement core functionalities	R/A	I	R/A	C
Conduct code reviews and refactoring	I	R/A	C	R/A
Commit and push code to the repository	R/A	C	I	C
Task 8: Backend Source Code				

Set up the backend infrastructure	C	R	A	C
Implement API endpoints and business logic	I	A	I	R
Conduct unit testing for backend components	R/A	C	R/A	C
Commit and push code to the repository	R/A	I	R/A	C
Task 9: Test Plan and Test Cases				
Define testing strategy and objectives	R/A	C	R	C
Execute testing and document results	R/A	I	C	R
Task 10: Cloud Infrastructure Setup				
Design cloud architecture	C	R/A	I	C
Set up cloud resources	I	R/A	C	I
Configure security settings	A	C	R	I
Test the cloud environment	R/A	C	C	C
Task 11: Mobile-Backend Integration Plan				
Define integration points	R	A	I	C
Conduct integration testing sessions	R/A	C	R/A	C
Finalize integration documentation	R/A	R/A	R/A	R/A
Task 12: Verification and Validation Document				

Define verification and validation criteria	R/A	R/A	C	I
Conduct reviews to ensure all requirements	I	R/A	C	R/A
Document verification and validation results	C	R/A	C	R/A
Review with advisor	I	R	A	I
Task 13: User Manual for API and Mobile App				
Draft user manuals	R/A	C	R/A	I
Finalize and publish user manual	R/A	R/A	R/A	R/A

7.3. Justification

In our project planning, we allocated enough time for each task based on its difficulty. We also arranged some tasks to overlap so we could work on easier ones at the same time. We built in breaks to regroup and discuss our plans.

By the time of our first release, we aim to complete the Project Estimation Report, Software Requirements Specification, Software Design Specification, API Documentation, Mobile App Wireframes and Design Mock-ups, and the Mobile App Source Code. In the following months, we will focus on the backend source code since it is the most challenging and time-consuming part of our project.

After completing the backend API, we will shift our focus to preparing the Verification and Validation Document and creating a User Manual for both the API and the mobile app. We believe our timeline is suitable for finishing the project with each deliverable completed on time.

8. Faculty Advisor Signature

This project proposal has been reviewed and approved by our faculty advisor.

Faculty Advisor Name: Dr. Yili Tang

Signature: _____

Date: 2024 Oct 9

9. Glossary

API (Application Programming Interface): A set of protocols and tools that allow different software applications to communicate with each other.

Autonomous Robots: Robots capable of performing tasks without human intervention, using sensors and algorithms for navigation.

Backend: The server-side component of a software system that processes requests, manages data, and communicates with other components.

Backend API: The server-side interface of an API that handles data processing, performs tasks, and sends responses to the client.

Battery Management: The process of monitoring and maintaining the battery levels of robots to ensure adequate power for operations.

CI/CD (Continuous Integration/Continuous Deployment): Development practices that involve regular code integration (CI) and automating software deployment (CD).

Cloud Infrastructure: A combination of hardware and software that provides computing services, like data storage and processing, via the internet.

Cloud Server: A server hosted online that offers scalable and on-demand computing resources for storing and processing data.

Collision Detection: A system or algorithm enabling robots to identify obstacles and avoid collisions during navigation.

Data Flow Diagram: A visual representation of data movement through a system, illustrating component interactions and processes.

Data Retrieval: The process of obtaining and displaying data, such as robot status or location, from a server or database.

Dijkstra's Algorithm: A pathfinding algorithm that finds the shortest paths from a starting node to other nodes in a graph.

Git: A version control system for tracking changes in source code during software development.

GitHub: An online platform for version control and code collaboration that hosts Git repositories.

Indoor Mapping: The creation of maps for indoor environments, allowing robots to navigate within buildings.

Integration Testing: Testing where individual components are combined and evaluated as a group to ensure proper interaction.

Latency: The delay between an action and the corresponding system response, often in data transmission.

LIDAR (Light Detection and Ranging): A technology using laser light to measure distances, aiding robots in detecting obstacles and navigating.

Mobile Application (App): Software for mobile devices that enables users to interact with and control robotic systems.

Navigation: The process of directing a robot's movement from one location to another within an environment.

Node.js: A runtime environment for executing JavaScript on the server side, commonly used in backend development.

Obstacle Avoidance: The ability of robots to detect and avoid obstacles while navigating their surroundings.

Operational Status: The current state of a robot, indicating whether it is active, idle, or experiencing an error.

Path Optimization: Techniques to find the most efficient routes for a robot, considering factors like distance and obstacles.

React Native: A framework for developing mobile applications using JavaScript and React, allowing cross-platform support.

Real-time Data Processing: Immediate processing of incoming data to provide up-to-date information and responses without delay.

RESTful API: An API style that uses HTTP requests to perform CRUD (Create, Read, Update, Delete) operations.

Robot Control System: Software enabling users to manage and direct the operations of a robot, including its movements.

ROS (Robot Operating System): An open-source framework for robotics software development, offering tools for robot control.

Scalability: The capacity of a system to handle growing workloads or data volumes effectively.

Sensor Fusion: The combination of data from multiple sensors to improve accuracy in navigation and obstacle detection.

Software Design Specification (SDS): A document outlining a software system's architecture, components, and data flow.

Software Requirements Specification (SRS): A document detailing a software system's functional and non-functional requirements.

Sprint: A time-boxed development iteration used in Agile methodologies for completing tasks.

TensorFlow: An open-source library for machine learning, often used for tasks like image recognition in robotics.

Unit Testing: Testing individual components or functions of a software system to ensure they work as intended.

User Acceptance Testing (UAT): The final phase of testing where real users validate that the system meets their requirements.

User Interface (UI): The part of an application that users interact with, allowing them to control and monitor the system.

User Experience (UX): The overall experience and satisfaction a user has while interacting with a system, focusing on ease of use and accessibility.

WebSocket: A communication protocol that provides real-time, full-duplex communication over a single TCP connection.

Wireframe: A basic visual guide that represents the layout of a user interface.