ÇANKAYA UNIVERSITY

FACULTY OF ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

Project Report

CENG 407

Innovative System Design and Development I

Sentinel: Autonomous Discovery Vehicle

Team Members:

Turgut Utku ALTINKAYA

Burak ATEŞ

Yunus Emre DİNÇEL

Bayram Alper KILIÇ

İlteriş SAMUR

Advisor: Assist. Prof. Dr. Serdar ARSLAN

Table of Contents

1. Introduction ... 1
2. Literature Review ... 2
 2.1. Introduction ... 2
 2.2. Autonomous Movement ... 2
  2.2.1. What is LIDAR? ... 3
  2.2.2. Object Detection ... 3
  2.2.3. Movement & Object Avoidance ... 4
 2.3. Path Planning ... 6
  2.3.1. Dijkstra’s Algorithm ... 6
  2.3.2. A* Algorithm ... 7
  2.3.3. D* Algorithm ... 7
  2.3.4. Rapidly-Exploring Random Tree ... 8
  2.3.5. Genetic Algorithm ... 9
  2.3.6. Ant Colony Optimization ... 9
  2.3.7. Firefly Algorithm ... 10
 2.4. 3D Mapping ... 10
  2.4.1. What is ROS? ... 11
  2.4.2. What is SLAM? ... 12
  2.4.3. Analysis of ROS based SLAM Algorithms ... 13
 2.5. Object Classification ... 16
  2.5.1. Object-Oriented Semantic Mapping ... 16
  2.5.2. YOLO (You Only Look Once) ... 18
 2.6. Related Works ... 19
  2.6.1. Sensor Fusion for Enhanced 3D Perception ... 19
  2.6.2. 3D-printed robot that uses SLAM for autonomous navigation ... 20
3. Software Requirement Specification ... 22
 3.1. Introduction ... 22
  3.1.1. Purpose of The Software Requirement Specification ... 22
  3.1.2. Scope of The Project ... 22
  3.1.3. Glossary ... 24
 3.2. Overall Description ... 26
  3.2.1. User Characteristics ... 26

3.2.2. Product Perspective 27
3.2.3. Product Functions 29
3.3. Requirement Specification 30
3.3.1. External Interface Requirements 30
3.3.2. Functional Requirements 36
3.3.3. Non-Functional Requirements 49
3.4. Use Cases 52
3.4.1. Movement 52
3.4.2. Data Stream 53
3.4.3. Manual Mapping 53
3.4.4. Autonomous Mapping 54
3.4.5. Object Detection 54
4. Software Design Document 55
4.1. Introduction 55
4.1.1. Purpose of this Document 55
4.1.2. Scope of this Document 55
4.1.3. Glossary 56
4.2. Overview of Software Design Document 57
4.3. System Design 58
4.3.1. Architectural Design 58
4.3.2. Data Flow Diagrams 59
4.3.3. Class Diagrams 64
4.3.4. Activity Diagrams 65
4.3.5. Sequence Diagrams 70
4.4. User Interfaces 74
4.4.1. Home Page 74
4.4.2. Admin Dashboard with 2D Map 75
4.4.3. Admin Dashboard with 3D Map 75
4.4.4. Interactive UI Link 76
5. Project Work Plan 76
6. Conclusion 76
References 77

1. Introduction

This project report prepared for The Sentinel: Discovery Autonomous Vehicle includes materials, contents, and working reports related to the Literature Review, Software Requirement Specification, and Software Design Document. Under the title of Literature Review, a preliminary working report was prepared mentioning the algorithms, approaches, and working mechanisms of the hardware components required for the project and related works used in the literature related to the titles as are necessary for the project (Movement, Path Planning, 3D Mapping, and Object Classification). The Software Requirement Specification (SRS), provides us with a detailed understanding of the project's capabilities in many aspects such as User Characteristics, Product Perspectives, Functions, Interfaces, and Functional & Non-Functional Requirements. In addition, Use Case diagrams aim to make the functional requirements easier to understand visually. This part covers the entire development lifecycle of Sentinel from concept to development. Software Design Document (SDD) provides a detailed presentation of the structure and architecture of the software in Sentinel. With this working report, various Designs are presented to users, partners, and stakeholders before the implementation phase of the project. This document includes the sequence diagram in which functions take action, and the Activity Diagram visualizing how functions will behave in which situations after user interaction with start and stop points. In addition, the Admin Dashboard in the User Interface section of the Sentinel project presents an interactive interface prototype. This project report, created by bringing together these three separate documents, provides information about the purpose and development stages of The Sentinel project.

2. Literature Review

2.1. Introduction

Autonomous robotic systems that can map and navigate unknown areas can be advantageous in various fields such as search and rescue, agricultural operations, and industrial automation. The aim of this project is to develop an autonomous vehicle that can perform 3D mapping and autonomous movement using Raspberry Pi, PiCamera, and LIDAR sensor. The two main components used in the development of this project are ROS (Robot Operating System) and SLAM (Simultaneous Localization and Mapping) and these are used for mapping purposes as well as enabling the localization of the vehicle. The role of Raspberry Pi in this case is to act as a publisher that sends the information obtained from the sensors and sends it to a remote computer. PiCamera and LIDAR sensors provide basic visual information and depth information about the environment for mapping and identification of obstacles.

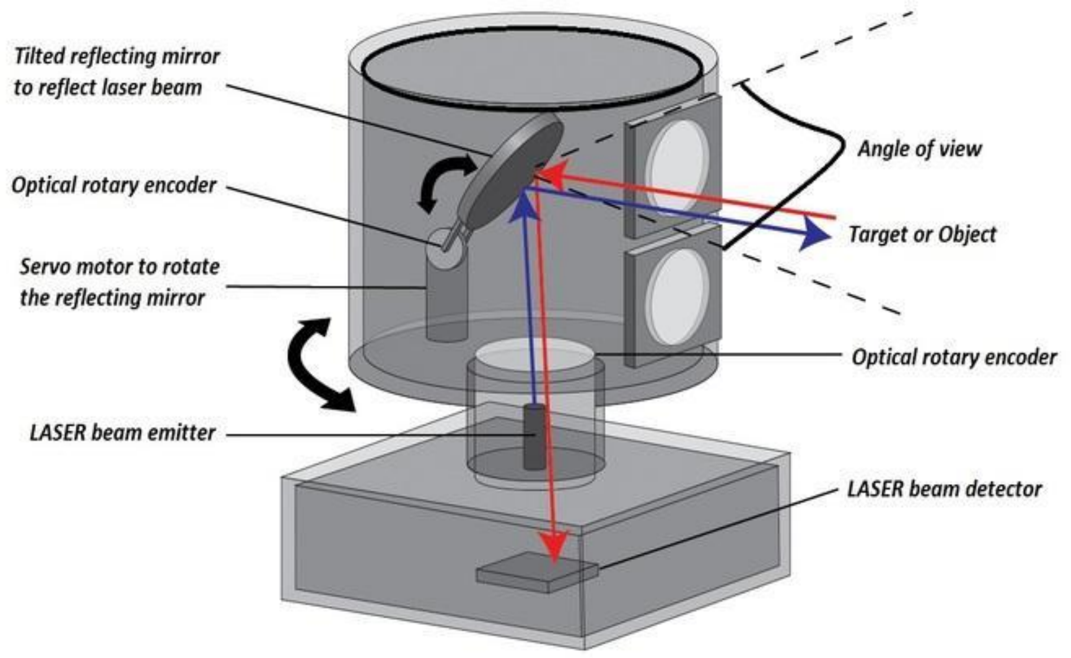
While the vehicle operates autonomously, its environment is mapped to a 3D model using an application named RTAB-Map. The generated map will be visible in real-time using RViz. After the generation of the 3D map, we will classify the objects that are found in that environment. This not only makes the environment clearer for the robot, but also provides a clear and active strategy in which autonomous systems can be implemented. The results of the project will provide evidence of the ability of Raspberry Pi, ROS, and SLAM to map an unknown environment in 3D, which can then be used in rescue operations, defense industry, etc.

2.2. Autonomous Movement

Autonomous vehicles rely on their perception systems to acquire information about their immediate surroundings. It is necessary to detect the presence of other vehicles, pedestrians and other relevant entities. Safety concerns and the need for accurate estimations have led to the introduction of Light Detection and Ranging (LIDAR) systems in complement to camera or radar based perception systems [1].

2.2.1. What is LIDAR?

The LIDAR (Light detection and ranging) utilizes the laser phenomenon to detect objects. The sensor emits laser pulses (aka light beams) to the surrounding environment, the reflected lights are captured by the sensor. After that, the light speed and time elapsed is used to calculate distance from the object by the LIDAR sensor [2].



2.2.2. Object Detection

The vehicle uses a LIDAR sensor to detect and map nearby obstacles by the LIDAR sensor attached to itself. This allows it to measure distances, creating a real-time 3D map. When an obstacle is detected, the vehicle's algorithms adjust its route, enabling it to navigate safely and avoid collisions, even in challenging environments.

2.2.3. Movement & Object Avoidance

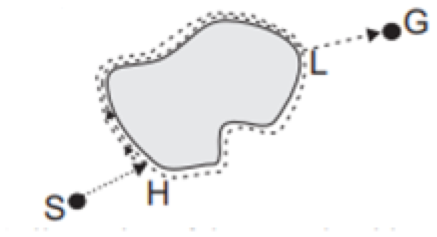
The vehicle can move and discover autonomously, but moving and detecting objects is not sufficient for an autonomous vehicle. It also needs to avoid the obstacles which are detected by the LIDAR. There are some algorithms which are used to avoid obstacles while reaching the endpoint.

2.2.3.1. Bug Algorithm

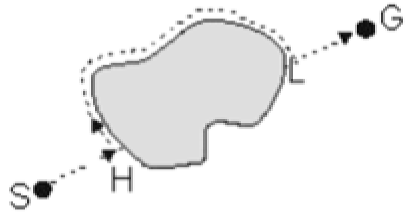
The Bug algorithm is a simple obstacle avoidance technique for robots navigating toward a target. When the robot encounters an obstacle, it follows the obstacle's boundary until it finds a point closer to the goal. It then resumes moving directly toward the target.

There are different variations, like Bug1 and Bug2:

- **Bug1:** The robot circles the obstacle entirely before deciding on the optimal exit point.



- **Bug2:** The robot follows the obstacle boundary only until it finds a point on the line between its start and the goal.



2.2.3.2. Probabilistic Road map (PRM) Algorithm

PRM is a simplified sample-based algorithm which works by constructing a probabilistic road map comprising networks of connected nodes in a given map based on free and occupied spaces to find an obstacle free path from start to end point. Once the roadmap is constructed, a path connecting source to destination is extracted using graph searching techniques [3].

2.2.3.2. Artificial Potential Fields

The artificial potential field method guides a robot by generating virtual forces that influence its movement. An attractive force pulls the robot toward its goal, while repulsive forces push it away from obstacles to prevent collisions. The robot's direction and speed are determined by the combined effect of these forces, allowing it to navigate smoothly. However, this method can struggle with "local minima," where attractive and repulsive forces balance out, potentially trapping the robot without a clear path to the goal [3].

2.2.3.2. Obstacle Avoiding Algorithms Comparison Table

Algorithm	**Advantage**	**Disadvantage**
Bug Algorithm	Simple and easy to implement. Effective in small, static environments.	Inefficient in complex or dynamic environments. Can get stuck in local minima.
PRM Algorithm	Fast, continuous path adjustment. Smooth, real-time navigation.	Can get stuck in local minima. Struggles with multiple obstacles creating conflicts.
APF	Works well for large, complex environments. Effective for global path planning.	Computationally expensive. Not ideal for real-time navigation in dynamic environments.

2.3. Path Planning

Path planning is a crucial component for autonomous vehicles to navigate efficiently and safely within diverse and complex environments. Autonomous vehicles rely on the path planning algorithms to create routes that avoid obstacles by minimizing travel cost like time, distance, and energy. The path planning algorithms are designed to meet the these requirements by generating optimal paths to the vehicle's environment, capabilities, and constraints. According to the environment, path planning approaches can be categorized into global and local methods. Global path planning algorithms are used when the environment is known. The algorithms compute a complete route from start to finish. The global path planning algorithms are suitable for static environments where the obstacles and boundaries are predictable. Local path planning algorithms, however, are essential in dynamic or unknown environments, where real-time adjustments must be made based on the sensor data as new obstacles or changes in the environment are detected.

An efficient path planning algorithm depends on factors like, computational resources, complexity of the environment and the need for real-time responsiveness. For static environments, classical approaches like Dijkstra's and A* are suitable since the environment is mapped well and can produce optimal paths with relatively low computation. Dynamic or unknown environment, on the other hand, more adaptive techniques like D*, RRT (Rapidly-Exploring Random Trees), Genetic Algorithm, ACO (Ant Colony Optimization) and Firefly algorithms are used [4].

2.3.1. Dijkstra's Algorithm

Dijkstra's algorithm is a classic graph search based algorithm to find the shortest path between node in a graph. The algorithm dives node into two sets, "visited" and "unvisited". The starting node assigned distance of zero, while others are set to infinity. Algorithm updates of each neighboring node if a shorter path is found. After updating neighbors, the current node moves to the "visited" set, and the process repeats until all nodes are visited [5]. Dijkstra's algorithm well perform on the static environments where the map is known in advance. However, the algorithm can be computationally expansive for large graphs in order to travel various nodes.

2.3.2. A* Algorithm

A* is an informed based algorithm used to finding the shortest path. It operates similarly to Dijkstra’s algorithm, but A* extends the algorithm by adding heuristics cost. A* algorithm evaluates node by calculating a function $f(n) = g(n) + h(n)$, where:

- $g(n)$ is the cost of path from start node, to the current node.
- $h(n)$ is the heuristic estimate of the cost of the current node to the goal node.

Despite Dijkstra's algorithm, A* algorithm prioritize nodes that are promising, guiding the search towards the goal while ensuring the path remains optimal and saving a significant amount of computation time [6].

2.3.3. D* Algorithm

The D* (Dynamic A*) algorithm is a path planning technique designed for dynamic or unknown environments where obstacles can appear or change as the autonomous vehicle navigates [7].Unlike, traditional algorithms like Dijkstra and A*, which are re-computes paths from scratch, D* adapts by updating only the parts of the path that are affected by changes. This localized update significantly enhances computational efficiency

The D* algorithm have operations which are “NEW”, “OPEN”, “CLOSED”, “RAISE”, and “LOWER” [8].

- NEW, meaning it has never been placed on the OPEN list.
- OPEN, meaning it is currently on the OPEN list.
- CLOSED, meaning it is no longer the OPEN list.
- RAISE, indicating its cost is higher than the last time it was on the OPEN list.
- LOWER, indicating its cost is lower than the last time it was on the OPEN list

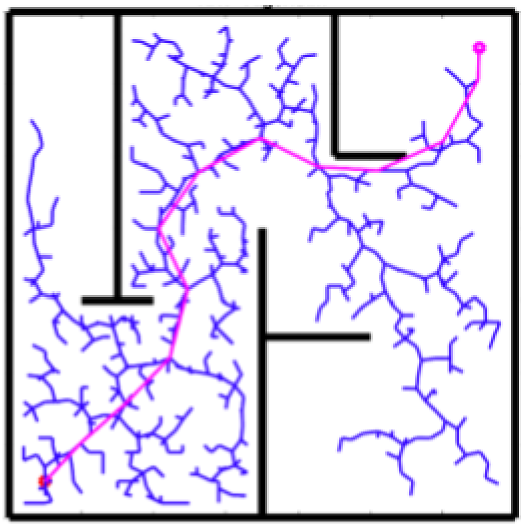
D* algorithm iteratively selects and evaluates nodes from the “OPEN” list, starting from the goal node and working backwards towards the start. Each expanded node has a back pointer to the next node towards the target, and it knows the exact cost to reach the goal. When an obstruction is detected along the path, affected nodes are added to “OPEN” list and marked as “RAISED”. Before increasing, the cost of a “RAISED” node, the algorithm checks its neighbors to see if the cost can be reduced. If not, “RAISED” state is propagated to its descendants, which are nodes with back pointers to it. These nodes are then evaluated, and the “RAISE” state continues to spread. If a “RAISE” node’s cost can be reduced, its back pointer is updated, and the “LOWER” state is passed to its neighbors [9].

2.3.4. Rapidly-Exploring Random Tree

The Rapidly-Exploring Random Tree (RRT) algorithm is a sampling based path planning algorithm method designed to efficiently search high-dimensional spaces, particularly for system with complex dynamics and constraints. RRT is used in autonomous vehicles in part of path planning due to its ability to explore large configuration spaces rapidly.

RRT operates by incrementally constructing a tree rooted at the initial configuration and expanding toward random samples within the search space. At each iteration, a new point is randomly selected, and the algorithm identifies the nearest node in the current tree. The tree then extends from this nearest node toward the sample point, constrained by the system's dynamics. This expansion is toward unexplored regions, allowing RRT to cover the configuration space effectively without extensive prior knowledge [4].

The advantages of the RRT algorithm lies in its speed and ease of implementation, It is relatively fast compared to other path planning algorithms. Its main cost is the search for closest neighbor, which becomes more computationally intensive as the number of generated vertices increases [10].



2.3.5. Genetic Algorithm

The Genetic Algorithm (GA) are adaptive heuristic search methods within evolutionary algorithms inspired by natural selection and genetics. These are intelligent exploitation of random searches, using historical data to guide the search toward higher performing regions in the solution space [11].

The GA process starts with a population of random paths, evaluated by a fitness function for criteria like path length, obstacle avoidance, or smoothness. Through selection, crossover and mutation, GA iteratively refines the population. Selection chooses high-fitness individuals as “parents” for the next generation, crossover combines parent traits to enhance diversity, and mutation introduces small random change to explore new areas in the search space.

The fitness function plays a critical role in directing the algorithm towards to optimal solutions. For path planning, it typically incorporates constraints like distance, obstacle avoidance, and path smoothness, with each factor weighted according to priority. Over successive generations, GA converges on a set of solutions that balance these factors, progressively refining the path quality.

2.3.6. Ant Colony Optimization

The Ant Colony Optimization (ACO) is a natural-inspired optimization algorithm. It is a suitable option for path planning, especially in complex, changing environments. Modeled on the natural behavior of ants locating the shortest path to food [12]. ACO operates by simulating “artificial ants” that traverse a graph from starting point to a target, searching for the shortest path while avoiding obstacles. As ants move, they deposit virtual “pheromones” on the edges of the graph, which act as probabilistic cues to guide subsequent ants. Over time, paths with higher pheromone concentrations attract more ants, reinforcing the desirability of shorter, more efficient routes. Conversely, pheromones on less-traveled paths cleared, allowing the algorithm to avoid local optima by reducing the likelihood of ants choosing suboptimal paths [4].

2.3.7. Firefly Algorithm

The Firefly Algorithm (FA) is metaheuristic optimization technique inspired by the communication and social behavior of fireflies. It has been applied efficiently to path planning for autonomous vehicle systems. The core principle of FA is that fireflies are attracted to brighter ones, moving towards those with higher intensity in order to optimize their path based on specific objectives like distance, safety, and smoothness [13].

The Firefly Algorithm (FA) helps autonomous vehicle systems navigate by balancing exploration with optimization. Each firefly (or solution) is attracted to better options, adapting paths dynamically for complex and unknown environments. FA's mix of randomness and directed movement enables effective handling of obstacles, while modifications like decreasing step size enhance convergence speed and efficiency, making FA suitable for real time autonomous vehicle [14].

2.4. 3D Mapping

3D mapping is a process in robotics that involves creating a three-dimensional representation of an environment. Unlike traditional 2D mapping, which only captures flat layouts, 3D mapping provides a full representation of the environment. 3D mapping is crucial in applications such as autonomous navigation, object detection, and obstacle avoidance, which require accuracy and a deep understanding of depth for effective robot operation. Robotic systems usually utilize LIDAR sensors and cameras to create 3D maps of environments. These sensors provide distance, depth, and visual data to robotic systems for constructing maps of the environment. In this context, ROS (Robot Operating System) is crucial. ROS provides frameworks and tools for integrating sensors and processing 3D mapping data. The most commonly used technique for this purpose is SLAM (Simultaneous Localization and Mapping). SLAM enables robots to build a 3D map of their environment while simultaneously tracking their own location within it. By utilizing SLAM algorithms, robots can navigate unknown spaces and adapt to changing environments. This combination of ROS and SLAM, along with data from LIDAR sensors and cameras, allows robotic systems to construct accurate and dynamic 3D maps essential for autonomous tasks.

2.4.1. What is ROS?

ROS (Robot Operating System) is an open-source robotics middleware suite. ROS is not an operating system. ROS is a set of software frameworks for robot software development. It provides tools for managing different hardwares and computers working together with hardware abstraction, low-level device control, message passing and hardware management.

ROS processes are represented as nodes in a graph structure connected by edges called topics. Nodes can represent various components in a robotic system such as sensors, services, actuators, and decision-making units. ROS nodes can pass messages to other ones through a publish-subscribe model. Nodes can publish messages to specific topics, and other parts of the system can subscribe to these topics to receive the data in real-time [14].

2.4.1.1. The Role of ROS in 3D Mapping

The ROS has an important role in 3D mapping by providing the frameworks for managing the SLAM algorithm. ROS facilitates the deployment and operation of SLAM algorithms through its middleware capabilities. Specifically, ROS enables the following key functions in 3D mapping:

Sensor Integration and Data Processing: ROS can handle and synchronize multiple data sources such as sensors and cameras. ROS will configure, process and forward the data to nodes that handle the SLAM algorithm.

SLAM Algorithms Frameworks: ROS provides the structured environment to implement various SLAM algorithms, such as GMapping, Hector SLAM, Karto SLAM, and RTAB-Map. These frameworks use sensor and camera data for building 3D maps with SLAM algorithms.

Simulation Environment: The ROS includes simulation frameworks like Gazebo. This simulation environment enables the testing of SLAM algorithms with virtual settings. This capability of ROS enables the replicate real-world conditions in simulation before deploying the hardware.

Loop Closure: Loop closure basically helps a robot understand that an object has already been visited, and therefore the robot will update its location and the map accordingly. The ROS mapping frameworks can detect loop closures [15].

2.4.2. What is SLAM?

Simultaneous Localization and Mapping (SLAM) was first coined as a term in a scientific paper presented at the 1995 International Robotics Research Symposium. The Simultaneous Localization and Mapping problem asks if it is possible for a robot to be placed at an unknown location in an unknown environment and for the robot to incrementally build a consistent map of this environment while simultaneously determining its location within this map.

The basis of the SLAM is Bayesian Probability, as it provides a way to update beliefs with respect to new information. In SLAM, the robot constantly refines its understanding of its position and the map as it gathers more observations. Bayesian probability helps by combining prior beliefs (the robot’s current estimate of where it and the landmarks are) with new evidence (sensor readings or movement data) to produce an updated belief. Bayes Theorem is used for calculate the posterior probability which is:

$$P(xk, m \mid Z0{:}k, U0{:}k, x0) = \frac{P(Z0{:}k \mid xk, m) \cdot P(xk, m \mid U0{:}k, x0}{P(Z0{:}k \mid U0{:}k, x0)}$$

where:

$P(xk.m \mid Z0{:}k.U0{:}k.x0)$ is the posterior: An updated, more refined belief of the robot’s position and the map after incorporating new observations.

$P(Z0{:}k \mid xk.m)$is the likelihood: The probability of observing the new sensor data if the current estimate were correct.

$P(xk.m \mid U0{:}k.x0)$is the prior: The initial probability of the robot’s state and map before incorporating the latest observations.

$P(Z0{:}k \mid U0{:}k, x0)$ is the evidence: The normalization factor that adjusts the overall probability distribution, ensuring that the updated beliefs about the robot's position and map are valid and sum to 1.

As well as The Bayes Probability, The Kalman Filter also helps SLAM to predict and correct the robot's estimates of its location and the map. The Kalman Filter works recursively therefore, The robot will update its estimates step by step as new data comes in, which is essential for real-time navigation. Kalman Filter will predict the new state of the robot (location and map) with respect to previous state and movement commands. When a robot observes a new landmark, the Kalman filter compares this observation with its prediction. If there's a difference between the prediction and the observation, the Kalman filter adjusts the estimate. Also the Kalman filter also maintains a covariance matrix that represents the uncertainty in the robot's estimate of both its location and the landmarks. This matrix grows or shrinks based on how much new information comes in [16].

2.4.3. Analysis of ROS based SLAM Algorithms

2.4.3.1. GMapping

GMapping is a 2D SLAM algorithm that utilizes Rao-Blackwellized Particle Filter (RBPF) to build maps with respect to LIDAR data and odometry input. The Rao-Blackwellized Particle Filter uses adaptive resampling technique. This helps to decrease the particle-depletion problem and computational complexity. The GMapping algorithm is designed in such a way that it integrates the current sensor data with the odometry motion model. This helps the particle filter's prediction step of uncertainty about the robot's location to decrease. GMapping is sensitive to odometry errors, which can affect map quality. It's not useful for challenging environments.

2.4.3.2. Hector SLAM

Hector SLAM is a SLAM algorithm which integrates laser scan matching feature with the 3D navigation method by the help of the inertial system which employs the EKF (Extended Kalman Filter). The algorithm computes a 3D state estimation of the robot's position and orientation using the Extended Kalman Filter. Hector SLAM can compute the 6 degrees of freedom of the AGVs position and orientation during motion and parallelly the high update rate laser-based 2D map. It is a SLAM algorithm which does not use the odometry data. Thus, the Hector SLAM has an advantage when used in environments which exhibit the pitch and roll characteristics.

2.4.3.3. Karto SLAM

Karto SLAM is a graph-based algorithm that incrementally builds maps by optimizing the robot’s current pose with respect to previously mapped areas. The major disadvantage with the Kato SLAM is that the complexity of computation is proportional to the number of landmarks. As the number of landmarks increases, the computational complexity also increases.

2.4.3.4. RTAB-Map

RTAB-Map is a SLAM algorithm which can be implemented with any type of sensor that provides depth information or three-dimensional information. The RTAB-Map is based on the RGB-D Graph SLAM. It can be implemented with 3D lasers, Stereo vision and even with RGB-Depth cameras. This method has a global loop closing detection technique in-built to compute the vehicle pose and an effective memory managing system. The map is saved as a structured graph with the robot's position and orientation and the captured image at that pose as nodes. The Map also contains the odometry data or the loop closing matrices as its edges. The major disadvantage is that the complexity of computation increases with the increase in the size of the map. The RTAB-Map SLAM can also be implemented without the need for odometry data.

SLAM	Sensors	Classification	Based On	Odometry Needed	Loop Closure
GMapping	2D Lidar	Particle filter	FastSLAM	YES	YES
Hector SLAM	2D Lidar	Kalman filter	EKF	NO	YES
Karto SLAM	2D Lidar	Optimization based SLAM	Graph SLAM	YES	YES
RTAB-Map	3D Lidar, camera	Optimization based SLAM	RGB-D Graph SLAM	NO	YES

In a study analyzing and evaluating various algorithms for autonomous ground vehicles, their performance was examined in both real-time and simulated environments. The research focused on aspects such as mapping accuracy, quality, and adaptability to complex settings. Among the algorithms studied, GMapping was identified as the most effective for generating maps in real-time and simulation environments with respect to SSIM. The SSIM is a metric that will compare the two maps which are of the same size, i.e., length and breadth. The SSIM is a metric which calculates the mean value, the variance and the covariance of the intensities of the cells obtained from the image provided [17]

SLAM	SSIM (Simulation vs Ground truth)	SSIM (Real-Time vs Ground Truth)
GMapping	0.84	0.83
Hector SLAM	0.81	0.78
Karto SLAM	0.75	0.77
RTAB-Map	0.81	0.76

2.5. Object Classification

Object Classification determines which specific objects are in an image or video. It labels these objects [18]. This technology, which is frequently faced in autonomous vehicles, plays an important role in decision-making to prevent dangerous situations by interpreting the data collected by the vehicle's sensors and cameras and labeling the objects it may encounter while driving. Object localization specifically tracks where objects are located in an image or video. This determines the location of any object within a piece of visual content. For example, for driving an autonomous car, pedestrians, other vehicles around, and traffic lights are of great importance in decision-making during autonomous movement.

With Autonomous Discovery Vehicles, unlike self-driving cars, object detection, and object classification can be used for different purposes. In common with self-driving vehicles, it is aimed at the discovery vehicle to detect dangerous objects and perform mapping and discovery tasks safely by using LIDAR and camera data. In addition, object detection is of great importance in the object-oriented semantic mapping approach on the 3D Map created using SLAM or other algorithms.

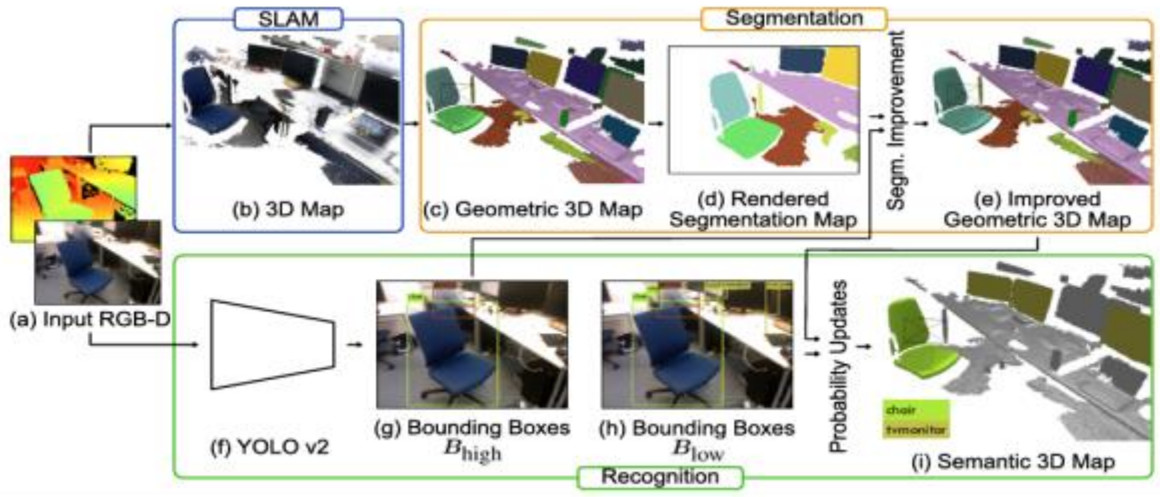
2.5.1. Object-Oriented Semantic Mapping

By using object detection methods to integrate semantic information on a 3D map created with the SLAM algorithm, the Object-Oriented Semantic Mapping approach, which provides a more advanced 3D map, makes it possible for Discovery Vehicles to make more meaningful mapping. This Semantic Mapping approach first geometrically segments incoming frames to create an initial 3D map, then overlays detected objects using bounding boxes, significantly enhancing object boundary definition.

Nakajima and Saito used various technologies and techniques in their approach to Object-Oriented Semantic Mapping [19]. Traditional semantic mapping systems, like those using CNN models, provide detailed scene understanding but often struggle with high computational costs, limiting real-time applicability in autonomous exploration. Nakajima and Saito's work provides an efficient alternative by combining object detection with incremental 3D map segmentation, reducing computational complexity without sacrificing accuracy. Also, this approach enhances real-time object-oriented segmentation and provides valuable insights into creating high-quality 3D maps suitable for real-world navigation and obstacle recognition.

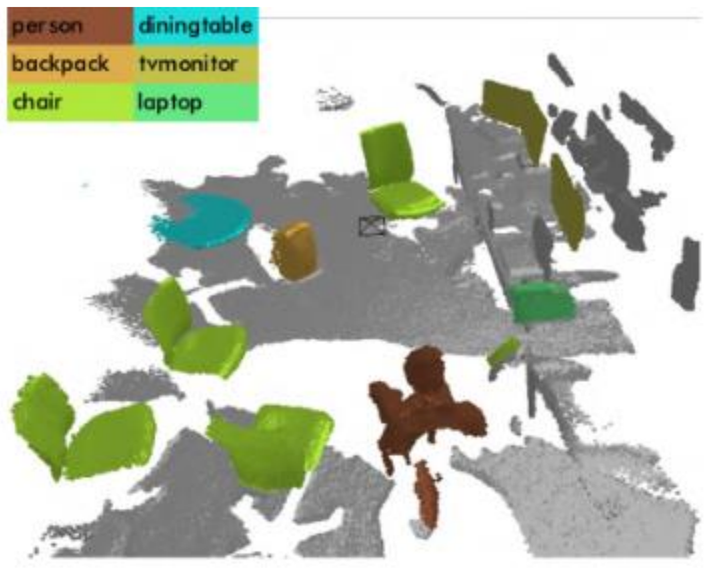
In this approach, incoming RGB-D frames are processed to estimate the camera pose via the Iterative Closest Point (ICP) and RGB alignment methods. This camera pose helps to accurately fuse new depth information into the 3D map, enabling a progressively built, dense representation of the scene. Another advantage of the approach is instead of applying 2D semantic segmentation directly to each frame, which can be slow, they use a geometric segmentation approach based on depth information. This technique groups geometrically similar areas into regions, which are incrementally segmented as new frames are processed.

By performing geometric segmentation on the 3D map, clear object boundaries can be established without needing prior object models. This method first organizes the 3D space into segments based on depth and shape features, creating a foundation for adding semantic labels. Also using the semantic information improves the segmentations. For instance, if two adjacent segments fall within the same bounding box and have similar semantic labels, they are merged. This process includes calculating a confidence score based on the probability that two segments represent the same object, thus refining the map accuracy and ensuring consistent labeling. To complete the object detection, YOLO outputs bounding boxes and class probabilities for objects detected in each frame. These bounding boxes are then used to assign semantic information to segments of the 3D map. After all this process the general view of this approach like in the Figure.



2.5.2. YOLO (You Only Look Once)

YOLO is one of the most popular model architectures and algorithms for object detection. The YOLO model uses one of the best neural network archetypes to produce high accuracy and overall speed of processing. This speed and accuracy are the main reason for its popularity [20]. YOLO divides an image into a grid and simultaneously predicts bounding boxes (the coordinates of potential objects) and class probabilities (what each object is likely to be) for each grid cell. This “one-shot” approach makes YOLO fast, and ideal for real-time applications. Each grid cell predicts multiple bounding boxes, each with a confidence score. This score represents the likelihood of an object within that box and the accuracy of the box's location. YOLO also assigns class probabilities to each detected bounding box, indicating what object class (e.g., person, car, chair) is most likely in that region. The architecture of YOLO allows the model to learn and develop an understanding of numerous objects more efficiently. Considering the Object-Oriented Semantic Mapping Results, YOLO effects can be easily observed with details in this
Figure.

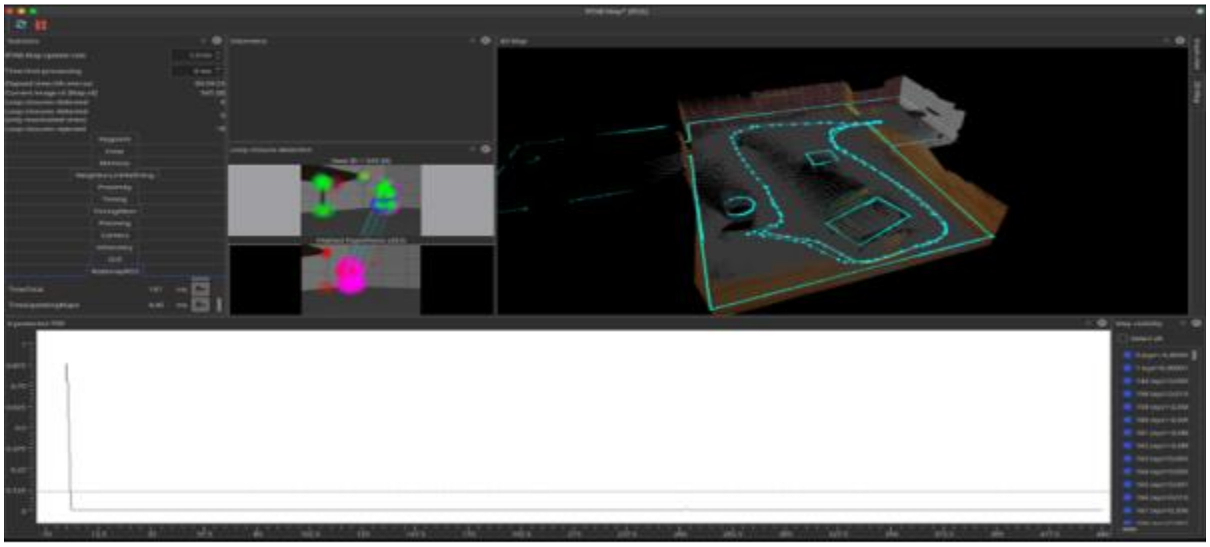


2.6. Related Works

The development of autonomous vehicles has led to profound advances in 3D object detection, map creation, and path finding algorithms. In our project, we aim to produce a vehicle that maps its surroundings in 3D and finds its way autonomously. Before implementing our project in practice, it is very important to read past studies and work on the subject. We can take these studies as examples when developing our project. In this section, related studies will be explained in detail.

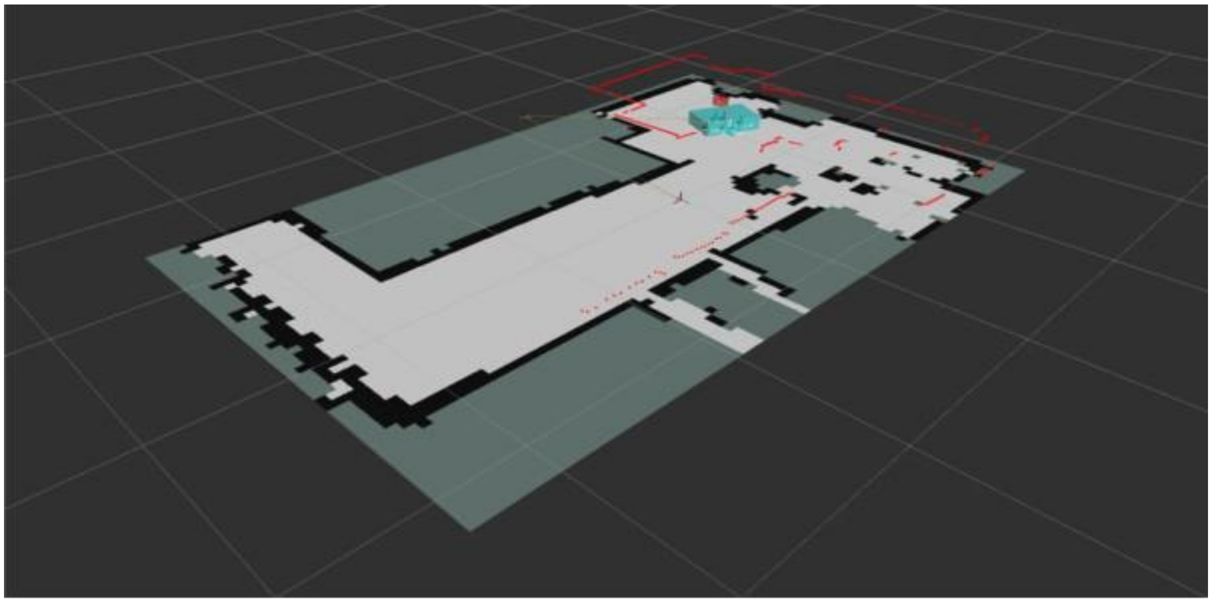
2.6.1. Sensor Fusion for Enhanced 3D Perception

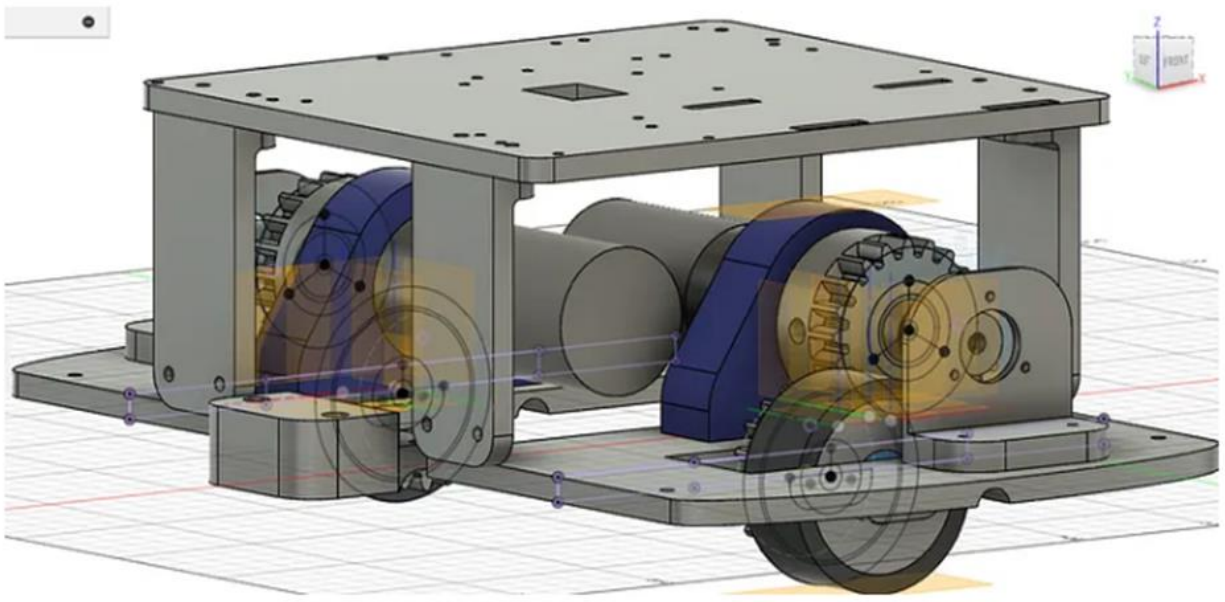
3D Maps are generated using the sensor fusion [21], which integrates LIDAR and camera data. The combination of LIDAR's depth data with the visual details from cameras provides a more comprehensive perception of surroundings. This technique not only addresses the limitations of the individual sensors, but also allows us to detect objects even under varying lighting and environmental conditions. An application named RTAB-MAP [22] (Real-Time Appearance-Based Mapping) incorporates sensor fusion algorithms to create detailed 3D maps for SLAM (Simultaneous Localization and Mapping). This application can be run using ROS [23] (Robot Operating System) or on IOS devices. For example, in a project named HURBA [24], they used RTAB-Map to generate a map for a robot simulation using ROS. The generated map can be displayed using RTAB-Map Viz (rviz) in real-time, as the robot generates the map when finding its way autonomously. This is a similar work that we plan to implement in our project. However, they only implemented the robot digitally, meaning that the robot is not present in the real life. Our project aims to implement it in real life, and also add some additional features like classifying the objects on the generated 3D map.



2.6.2. 3D-printed robot that uses SLAM for autonomous navigation

Another example project is a 3D-printed robot [25] that uses the SLAM (Simultaneous Localization and Mapping) algorithm to navigate autonomously. This Project also uses ROS and LIDAR to map its surroundings and find its way. However, the project only maps the environment in 2D, which means that objects inside that environment are not detected and classified. In ROS, there can be many nodes where users can write and run scripts, and data can be sent to other ROS applications via publisher nodes. Subscriber nodes can receive messages sent from publisher nodes, so that data from the publisher, which is the robot, can be processed on the remote desktop. In this project, instead of Raspberry Pi, they used Nvidia Jetson Nano, which is more expensive and powerful than Raspberry Pi. Since it is more powerful, they processed the data on that robot and did not send it through publisher nodes. In our project, we aim to send the data to a remote computer so that the data can be processed more efficiently and thus reduce the stress on the robot. The parts of the SLAM configuration with ROS are explained in detail on the project website. For example, the localization and mapping algorithm mainly uses three core frames. They are map, odom and base_link. Map is the frame that represents the real-world environment where the robot should be fixed; odom is the frame that remains fixed relative to the robot's initial position and is used to depict the robot's odometry location estimate. base_link is the frame that represents the robot's location. The map to odom transformation is calculated by the localization node, which mainly depicts the alignment between the sensor data and the incorrect odometry data over time [26]. In this project, they implemented the odom to baselink transformation, representing the difference between the estimated current position and orientation and the real ones.





This project is really similar to our project. However, since it does not create a 3D Map, we plan to further develop this project, add a 3D mapping using sensor fusion with LIDAR and camera data, and classify the objects detected in that environment while the robot autonomously explores and finds its way.

3. Software Requirement Specification

3.1. Introduction

3.1.1. Purpose of The Software Requirement Specification

The purpose of this document is to provide a detailed understanding of the scope, capabilities, functions and hardware of Sentinel. It plays an important role as it is a reference for the development team, stakeholders and other organizations involved in the development and improvement of Sentinel. This document covers the entire development lifecycle of Sentinel from concept to development. It provides a detailed description of the objectives, user interactions, vehicle motion, environment mapping and system architecture for the successful physical implementation and development of this vehicle.

3.1.2. Scope of The Project

The Sentinel idea stemmed from the development of unmanned vehicle technologies in modern warfare. In today's world, many countries use technology in the defense industry for both self-defense and operational purposes. When these vehicles are not used, they cause many soldiers to lose their lives. For this reason, there is a great need for unmanned vehicles in war zones. For this reason, many countries are investing in these vehicles. This situation caught our interest and we decided to make an autonomous environment mapping vehicle. However, our vehicle is a proof of concept and isn't suitable for the war zone.

The main purpose of Sentinel is to create 2D and 3D maps of its surroundings by moving autonomously. The vehicle can be sent to dangerous places to autonomously map the environment, thus preventing personnel loss. It can be used to create a map of the environment in many areas such as buildings that are not safe to enter, war zones and places contaminated with harmful gases. The importance of this is that it gives an idea about the place without human interaction and provides detailed 3D environment mapping that shows dangers in advance. It can also be used to detect life in rescue operations when a thermal camera is used.

The vehicle is mainly equipped with Raspberry Pi5, which runs on Ubuntu, YDLidar X2, and Picamera 3. The data collected from the lidar sensor and camera are combined and used to generate the 3D map utilizing the rtabmap_ros [27], a package used to generate a 3D point of clouds of the environment and/or to create a 2D occupancy grid map for navigation. It will mainly use the Robot Operating System (ROS) to control both autonomous and manual movement, and environment mapping.

The collected data is planned to be processed at a remote computer that is also equipped with ROS, and only the movement command from the remote computer will be returned to Sentinel. In this way, we also plan to provide a manual control mechanism which allows our users to interact with the vehicle any given time. Moreover, the generated map and the camera footage will be available to the user at the remote computer in real-time.

After the map has been generated, we plan to assign jobs to the Sentinel, such as finding the red chair from the generated map, and it will find that red chair. To achieve this, we also plan to detect the objects using YOLO [28], a library for object detection, and then store these informations using rosbag.

In summary, users can control the movement of the vehicle, view images from the vehicle, and view 2D and 3D maps in real time. All of this is done on a remote computer. On the other hand, Sentinel is responsible for sending sensor and camera data to the remote computer and obeying the command received from the remote computer. In the following sections of this SRS report, these functionalities and the hardware used for the vehicle will be explained in detail.

3.1.3. Glossary

Term	**Description**
Bags	A bag is a file format in ROS for storing ROS message data.
C++	Programming Language
DC Motor	A DC motor is an electrical motor that uses direct current (DC) to produce mechanical force. The most common types rely on magnetic forces produced by currents in the coils.
DSI	The Display Serial Interface (DSI) is a specification by the Mobile Industry Processor Interface (MIPI) Alliance aimed at reducing the cost of display controllers in a mobile device.
FPC	FPC connectors have been established in response to challenges in this emerging industry which calls for smaller centerline or timer distances, smaller capacity heights, and lightweight interconnection solutions as the industry trends towards miniaturization.
HDR	High dynamic range (HDR), also known as wide dynamic range, extended dynamic range, or expanded dynamic range, is a signal with a higher dynamic range than usual.
L298N Motor Driver	The L298N motor driver is a versatile module used to control both the speed and direction of DC motors.
Li-ion Battery	Li-ion battery is a type of rechargeable battery that uses the reversible intercalation of Li+ ions into electronically conducting solids to store energy.
LiDAR	LiDAR is short for Light Detection and Ranging. In LiDAR, laser light is sent from a source (emitter) and reflected off objects in the scene.
NVMe	NVM Express (NVMe) is an open, logical-device interface specification for accessing a computer's non-volatile storage media usually attached via the PCI Express bus.
Pi Camera Module 3	Compact camera from Raspberry Pi
Powerbank	A power bank or battery bank is a portable device that stores energy in its battery. Power banks are made in various sizes and typically based on lithium-ion batteries
Publisher/Subscriber	Publish/subscribe is a messaging pattern where publishers categorize messages into classes that are received by subscribers.

Python	Programming Language
RAM	Random-access memory (RAM) is a form of electronic computer memory that can be read and changed in any order, typically used to store working data and machine code.
Raspberry Pi 5	Small single-board computer
ROS	Robot Operating System (ROS or ros) is an open-source robotics middleware suite that helps you build robot applications.
ROS Topic	Buses over which nodes exchange messages
Rosbag	A set of tools for recording and playing back to ROS topics.
RPM	RPM stands for revolutions per minute and is a measure of how fast the engine is spinning.
RTAB-Map	ROS framework for 3D and 2D mapping.
SD Card	Secure Digital, officially abbreviated as SD, is a proprietary, non-volatile, flash memory card format the SD Association (SDA) developed for use in portable devices.
SLAM	Simultaneous Localization and Mapping. Algorithm for mapping. Build a map and localize your vehicle in that map at the same time.
SSD	A solid-state drive (SSD) is a type of solid-state storage device that uses integrated circuits to store data persistently.
The Sentinel	The Sentinel is a Discovery Vehicle. The name of our project, The Sentinel, is an autonomous vehicle with features such as 3D mapping and object detection.
Ubuntu	Linux based operating system
YDLidar X2	360-degree two-dimensional rangefinder sensor.
YOLO	You Look Only Once. Open source library that helps you object detection.

3.2. Overall Description

3.2.1. User Characteristics

3.2.1.1. Operator

The Operator is responsible for monitoring and controlling The Sentinel's operations. Their tasks include both manual and autonomous movement. Operators can use tools such as a keyboard or joystick for manual control, or manage its autonomous functions through a user interface. This type of user must have a basic understanding of The Sentinel's functionalities, as well as the minimum technical expertise required for routine tasks.

3.2.1.2. Developers and Engineers

The developers and engineers are responsible for designing, implementing, and maintaining both the hardware and software systems of The Sentinel. This also includes working with advanced technologies like ROS (Robot Operating System), mapping and path planning algorithms, and data processing frameworks. The developers and engineers should be capable of performing troubleshooting, optimizing the system, and making sure it works reliably in a variety of conditions. Moreover, they might be required in deploying updates, enhancing performance, or resolving various kinds of technical challenges that may occur while the system is operational.

3.2.2. Product Perspective

The Sentinel includes different hardware and software systems. The Sentinel Project aimed to produce a vehicle prototype with the specified features. So we used hardware tools for the prototype. On the software part Sentinel includes different computer science concepts: mapping algorithms (SLAM), path planning algorithms, object detection algorithms and also communication protocols. These algorithms will be implemented by different programming languages: C++, Python. Additionally, we use ROS2 and its frameworks like RTAB-Mapfor implementing these features. On the hardware part since our project is a vehicle, we used 4 wheels and 4 motors. The 2 motors on the right and the 2 motors on the left are connected in series with each other. Motors are powered by 8 x AA batteries with the voltage of 12V. The L298N motor driver is used to controlling motors for forward, backward left and right movement. The brain for the vehicle is Raspberry Pi 5. Raspberry Pi 5 is responsible for gathering visual data from Pi Camera Module 3 and distance data from YDLidar X2 which both are mounted on Raspberry Pi. When this data gathering Raspberry Pi real time streams both visual and distance data to a remote computer from ROS2 publisher/subscriber protocol. Remote computers are responsible for computations like mapping, object detection and path planning. For autonomous driving, a remote computer sends movement commands from the output of computations to Raspberry Pi and at Raspberry Pi this command is executed according to the input received. The detailed explanation of the hardware interface will be given at 3.1.2. Hardware Interfaces section and software interface will be given at 3.1.1.

In the below table, we stated what are the features of our vehicle, summarized explanation, and which tools we will use as hardware.

Task	Definition	Hardware Tools
Manual Driving	The Sentinel shall operate by a user from a remote computer with a joystick or keyboard.	● Raspberry Pi 5 ● Remote Computer ● Joystick or Keyboard ● 4 x motor ● Vehicle Chassis & tires ● L298N Motor Driver
Autonomous Driving	The Sentinel shall operate by itself while considering the ROS2 outputs.	● Raspberry Pi 5 ● Remote Computer ● YDLidar X2 ● 4 x motor ● Vehicle Chassis & tires ● L298N Motor Driver
3D Mapping	The Sentinel shall autonomously map unknown environments using ROS2's framework RTAB-Map.	● Raspberry Pi 5 ● Remote Computer ● YDLidar X2 ● Pi Camera Module 3
Object Detection	The Sentinel shall classify the objects at unknown environments.	● Raspberry Pi 5 ● Remote Computer ● Pi Camera Module 3
Real Time Data Stream	The Sentinel shall stream gathered camera and lidar data to remote computer and web applications.	● Raspberry Pi 5 ● YDLidar X2 ● Pi Camera Module 3

3.2.3. Product Functions

Manuel Movement: The system is designed to performs directional movements based on user inputs (left, right, up, down) either from joystick or a keyboard from joystick or keyboard.

Autonomous Movement: The system is designed to perform movement according to the system output from ROS2. The ROS2 output includes direction and speed. The system can handle the speed with the motor rpm rated from 0-100, adjusting the motor rpm according to the incoming speed value and providing the speed correctly.

Data Collection: The system is equipped to collect visual data in the form of images and videos using the Pi Camera Module 3, which is integrated to Raspberry Pi 5 hardware. Additionally it can gather comprehensive 360-degree distance measurement using the YDLidar X2 sensor, which operates with ROS2.

Real Time Data Transfer: The system enables real-time transfer of camera and LiDAR data using the ROS2 publisher/subscriber protocol. The data collected on the Raspberry Pi is transmitted to the remote computer efficiently, utilizing ROS2's message framework. Both the Raspberry Pi and the remote computer run ROS2 to ensure communication.

2D Mapping: The system can create a real-time 2D grid map of an unknown environment using distance and angle data from a LiDAR sensor. It employs the RTAB-Map framework, integrated within ROS2, to generate the 2D grid. RTAB-Map constructs the grid by considering the vehicle's starting point and the incoming distance measurements. Additionally, it detects loop closures (instances where the vehicle revisits a previously mapped location) and resolves them to prevent mapping errors or confusion.

3D Mapping: The system can create a 3D map of the environment using RTAB-Map, similar to how 2D mapping is performed. However, 3D mapping includes additional considerations. During the mapping process, image data from a camera integrated with a LiDAR sensor is placed on the map based on distance and angle. Unlike 2D mapping, which primarily considers distances along the x-axis, 3D mapping also incorporates distances along the y-axis, resulting in a three-dimensional representation of the environment.

Object Detection: The system can perform real-time object detection in an unknown environment using a Pi Camera Module 3 mounted on the vehicle. It utilizes a deep learning-based object detection framework YOLO (You Only Look Once), integrated into ROS2 to identify and classify objects in the surroundings. The system processes the camera's video feed to detect objects considering their position and size in the environment.

3.3. Requirement Specification

3.3.1. External Interface Requirements

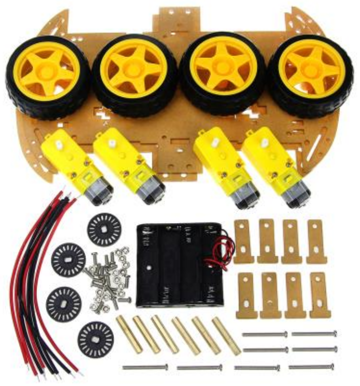
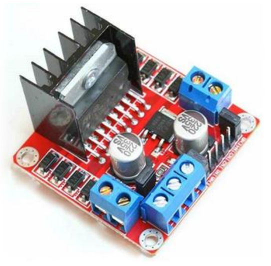
3.3.1.1. User Interface

The Sentinel application is used for the web. The Sentinel's user interface should be clear and understandable in English. In this context, as the Discovery Vehicle advances within the area it is exploring in The Sentinel User Interface, the images obtained from the camera and the 2D and 3D maps created with various algorithms will lively be presented to the user. With its simple, sustainable, easy-to-use interface, Sentinel will be able to offer various functions to the user. It offers many features to the users such as being able to go to the marked point on the map, being able to go to the location of the object by visualizing the objects detected with the camera on the map, and being able to view past mappings on user history. Users can create his/her account to access The Sentinel’s features easily. In the interface section, users can prevent the camera image from appearing with the live stream feature, and can only enable the map display feature. They can also experience the mapping features of the Sentinel Discovery Vehicle by choosing between manual mapping and autonomous mapping options. In addition to the live camera image during mapping, distance and proximity data from Lidar are also presented to users in the web interface. It can also be observed how close the discovery vehicle gets to an object.

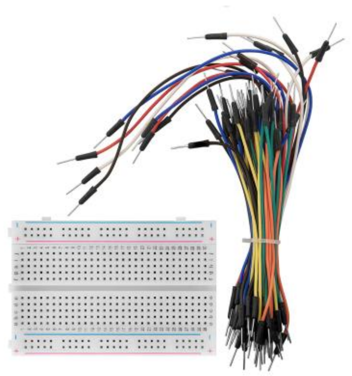
3.3.1.2. Hardware Interface

The Sentinel requires many separate components in the hardware section. Many hardware requirements for the use of features such as autonomous movement, object detection, and 2D - 3D Mapping are as in the table below. With these products, The Sentinel Autonomous Discovery Vehicle emerges. The intended uses, product images, and descriptions are also included.

Figure	Product Name	Purpose of U
	Raspberry Pi 5	Raspberry Pi is a cheap, small computer that has 8 GB RAM that runs Linux (various distributions), but also provides a series of GPIO (general purpose input/output) pins that allow you to control electronic components for physical computing. The Sentinel uses the Raspberry Pi for remote manual control, communication with the remote computer via ROS (Robot Operating System), and sending the image captured via the PiCamera and Lidar data to the remote computer for processing.
	LiDAR	LiDAR is short for Light Detection and Ranging. In LiDAR, laser light is sent from a source (emitter) and reflected off objects in the scene. LiDAR is an optical technology that is often cited as a key method for range sensing for autonomous vehicles. The Sentinel uses the LiDAR sensor to provide users with features such as autonomous movement, obstacle detection, and 3D Mapping.
	Pi Camera Module 3	Pi Camera Module 3, is the most popular and latest in the mainstream Raspberry Pi camera class. It has 12 MP and provides good low-light photography and output thanks to its improved low-light intensity. This module also supports High Dynamic Range (HDR), which makes it easier to get a sharp image

		output. The Sentinel processes the images obtained by the Pi Camera for object detection and displays the detected objects on 3D mapping.
	Vehicle Chassis Kit	The vehicle chassis kit forms the foundation and body of The Sentinel. There are 4 separate 6V parallel-connected motors for 4 wheels. It is 2-story with an extra chassis piece, lidar, and camera on the top. Powerbank, Motor driver, Raspberry Pi 5, breadboard, and cables are also located on the chassis. There are also 2 battery slots for the batteries needed to power the wheels.
	L298N Motor Driver	The L298N motor driver is a versatile module used to control both the speed and direction of DC motors. Used this driver to provide precise motion and control for The Sentinel, powered by the Raspberry Pi 5. The L298N can drive up to four DC motors, making it an ideal choice for our system's requirements.
	Power Bank	The Sentinel requires the use of a power bank to power the Raspberry Pi. The power bank, which provides 5V 5A output, provides enough power to power the Raspberry Pi.

	NVMe 2.0 500 GB SSD	To provide data flow and communication via Robot Operating System (ROS), ROS must also be installed on Raspberry Pi 5. Since SD cards are insufficient in speed and storage, the SSD with NVMe technology, which can transmit data at high speed with 500 GB storage, is used in The Sentinel. SSD is also of great importance for using the GUI of Raspberry Pi.
	12V Lithium-ion Battery	For the system to work properly, the motors must be given sufficient power. In this way, The Sentinel can move with the energy given to the wheels. For this purpose, sufficient power can be provided with a 12V Lithium-ion Battery. Since it is rechargeable, it contributes to recycling.
	Raspberry Pi Display Cable	With the Raspberry Pi Display Cable, you can physically connect the Pi Camera Module 3 and Raspberry Pi 5 and establish a connection between them. Technically Shielded cable to connect a DSI display to the 22-way FPC connector on Raspberry Pi 5.

	External Wires, Connectors, and Breadboard	Breadboard, external cables, and connectors are needed for many hardware operations such as connections between Raspberry Pi and motors, opening and closing motor connections, etc.
	Raspberry Pi M.2 HAT+	To use Raspberry Pi effectively and speed it up, we need an M.2 NVMe HAT to connect the NVMe SSD we use to Raspberry Pi. External drives can be associated with M.2 HAT, accelerating data transfer.

3.3.1.3. Software Interface

The Sentinel uses a client-server architecture. The client runs on the Raspberry Pi 5 and uses Ubuntu as the operating system to support the ROS environment. The server runs on a remote computer. This architecture facilitates communication between the Raspberry Pi and the remote computer to control system movement based on camera and sensor data.

3.3.1.3.1. Sensor and Camera Interface

The Raspberry Pi, as the client, is responsible for capturing the required data, such as camera and sensor information. The captured data will be transmitted to the server for processing. As an input, the client will capture data at a constant frequency and stream it to the server. The sensor data will be in a structured format compatible with ROS messages. The camera data will be sent to the server after being compressed to reduce the image size and facilitate faster communication.

3.3.1.3.2. Movement and Control Interface

The Sentinel supports both manual and autonomous movement. The movement control system is handled by ROS on the server. Manual movement includes two control systems: keyboard and joystick. In the keyboard system, when certain keys are pressed, the server sends the appropriate movement messages, such as 'Forward.' Similarly, in the joystick system, movement messages are sent, but the joystick also includes a coefficient that specifies how far the system will move. For autonomous movement, the server uses ROS to process the sensor and camera data, plan the best path for movement, and send the appropriate movement message to the client to move.

3.3.1.3.3. ROS Node Architecture

Both server and client will run on separate ROS nodes that interact with each other through publisher/subscriber architecture.

Client:

- **Sensor Nodes**: Collects the necessary sensor and camera data and publishes it to the ROS topic.
- **Movement Nodes:** Receive movement commands (either from manual and autonomous) and send signals to the vehicle's actuators.

Server:

- **Sensor Processing Nodes**: Process the camera and sensor data to detect obstacles, identify the vehicle environment.
- **Path Planning Nodes**: Uses the path planning algorithms to optimize the movement of the vehicle.
- **Control Nodes**: Sends movement instructions to client based on the processed data and planned path.

3.3.1.4. Communication Interface

The remote server and the vehicle are communicating with Wi-Fi technology. They must be on the same network. They will send and receive messages with the ROS publisher-subscriber model.

The publisher-subscriber model is a way of communicating in ROS (Robot Operating System) that helps share information between nodes using messages. In this model, a broadcaster allows multiple nodes to exchange data, such as sensor readings, control signals, or other information, by grouping it under a specific topic. For example, one node can send sensor data (publish), which other nodes can receive and use (subscribe).

A subscriber is like a listener that tunes in to a specific topic. It receives the messages broadcasted on that topic and processes the data to make decisions or control devices. Topics act as communication channels with names where publishers send messages and subscribers receive them.

Messages are structured pieces of data containing fields, like numbers or text, based on the application's needs. This model is important because it keeps publishers and subscribers independent. They can work without knowing about each other, which makes the system more flexible and modular. Multiple publishers and subscribers can communicate over the same topic without interfering with each other [29].

3.3.2. Functional Requirements

Use Case Numbers	SRS/4.1
Use Case Name	Get Movement Command
Related Use Cases	Set Movement Command
Description	Sentinel gets the movement command from Remote Computer System or User at the Remote Computer
Inputs	Movement Command
Source	Remote Computer System
Precondition	Connection between the Remote Computer System and Sentinel must be stable
Postcondition	Sentinel gets the movement command from the Remote Computer System
Scenario	1. The Remote Computer System or the User at the Remote Computer System sends the movement command 2. Sentinel gets the movement command
Exceptional Situations & Alternative Flows	Connection between the Remote Computer System and Sentinel might be lost, Sentinel stops until the connection is re-established

Use Case Numbers	SRS/4.1
Use Case Name	Change Movement Mode
Related Use Cases	Keyboard Movement, Joystick Movement, Set Movement Command
Description	In order for the Sentinel to move autonomously or manually, its movement mode must be changed.
Inputs	Movement Mode
Source	Remote Computer System
Precondition	Connection between the Remote Computer System and Sentinel must be stable
Postcondition	Remote Computer System changes the Sentinel's movement mode
Scenario	<u>Scenario 1:</u> 1. Initially Movement Mode is set to autonomous 2. If user at the Remote Computer wants to move the Sentinel manually, user changes the movement mode 3. Sentinel's movement mode has been set to manual <u>Scenario 2:</u> 1. Initially Movement Mode is set to manual 2. If user at the Remote Computer wants Sentinel to move autonomously, user changes the movement mode 3. Sentinel's movement mode has been set to autonomous
Exceptional Situations & Alternative Flows	Connection between the Remote Computer System and Sentinel might be lost, Sentinel stops until the connection is re-established

Use Case Numbers	SRS/4.2, SRS/4.4
Use Case Name	Collect Camera and Sensor Data
Related Use Cases	-
Description	System collects data from camera and sensors that are mounted on the vehicle.
Inputs	Camera and Sensor Data.
Source	Camera and Sensor hardware.
Precondition	Camera and sensors must be properly mounted on vehicle hardware
Postcondition	The vehicle system acquire camera and sensor data
Scenario	1. Camera and sensors starts to work 2. Vehicle system collects the data
Exceptional Situations & Alternative Flows	If there is a hardware problem between the camera or sensor hardware and the Raspberry Pi, data acquisition stops.

Use Case Numbers	SRS/4.1, SRS/4.2, SRS/4.3, SRS/4.4, SRS/4.5
Use Case Name	Publish Data to ROS Node
Related Use Cases	-
Description	System publishes collected camera and sensor data to a ROS node.
Inputs	Camera and Sensor Data.
Source	Vehicle System
Precondition	Camera and sensor data must be acquired.
Postcondition	Camera and sensor data published from ROS Node
Scenario	1. System publishes the data to the ROS node while new data comes.
Exceptional Situations & Alternative Flows	Connection between the Remote Computer System and Sentinel might be lost, The data transfer stops until the connection is re-established between Remote Computer System and Sentinel.

Use Case Numbers	SRS/4.1, SRS/4.2, SRS/4.3, SRS/4.4, SRS/4.5
Use Case Name	Get Data From ROS Node
Related Use Cases	-
Description	The remote computer must get published data from the ROS node.
Inputs	-
Source	ROS Publisher Node
Precondition	The camera and sensor data must be published to the ROS node.
Postcondition	The remote computer acquires camera and sensor data.
Scenario	1. The remote computer get data from the ROS node while new data publishes
Exceptional Situations & Alternative Flows	Connection between the Remote Computer System and Sentinel might be lost, The data transfer stops until the connection is re-established between Remote Computer System and Sentinel.

Use Case Numbers	SRS/4.2
Use Case Name	Stream Data to Web
Related Use Cases	-
Description	Remote Computer streams data to web applications.
Inputs	Camera and Sensor Data.
Source	Remote Computer System
Precondition	Camera and sensors are must gathered from ROS node./
Postcondition	The web applications acquires the camera and sensor data
Scenario	1. UDP sockets are opened between remote computers and web applications. 2. Remote computer sends the camera and sensor data through sockets.
Exceptional Situations & Alternative Flows	Connection between the Remote Computer System and Sentinel might be lost, The data transfer stops until the connection is re-established between Remote Computer System and Web

Use Case Numbers	SRS/4.2
Use Case Name	Render Page
Related Use Cases	-
Description	Web application renders the page for showing new data.
Inputs	Latest camera and sensor data.
Source	UDP Socket between remote computer and web application.
Precondition	Camera and sensor data are must be sent from remote computer to web application
Postcondition	New data can be seen from the web page.
Scenario	1. Web application acquires data from sockets. 2. Renders the page for showing the last data.
Exceptional Situations & Alternative Flows	The web application may stop working and in this case the latest data cannot be seen.

Use Case Numbers	SRS/4.1, SRS/4.3
Use Case Name	Keyboard Movement
Related Use Cases	Get Movement Command, Change Movement Mode
Description	User sends movement data to the server with a keyboard.
Inputs	Keystroke
Source	User
Precondition	Server must be on and connected to the working car.
Postcondition	Sentinel will move into the desired direction.
Scenario	1. User will press one of the movement keys on keyboard (w,a,s,d) 2. Server will recognize pressed key
Exceptional Situations & Alternative Flows	Connection between the Remote Computer System and Sentinel might be lost, Sentinel stops until the connection is re-established

Use Case Numbers	SRS/4.1, SRS/4.3
Use Case Name	Joystick Movement
Related Use Cases	Get Movement Command, Change Movement Mode
Description	User sends movement data to the server with a joystick.
Inputs	Joystick Movements
Source	User
Precondition	Server must be on and connected to the working car.
Postcondition	Sentinel will move into the desired direction.
Scenario	1. User will use joystick to move car 2. Server will recognize movement direction and speed
Exceptional Situations & Alternative Flows	Connection between the Remote Computer System and Sentinel might be lost, Sentinel stops until the connection is re-established

Use Case Numbers	SRS/4.1, SRS/4.3
Use Case Name	Set Movement Command
Related Use Cases	Get Movement Command, Change Movement Mode
Description	Sets current movement mode of the Sentinel
Inputs	Movement data
Source	User
Precondition	Server must be on and connected to the working car.
Postcondition	Sentinel will move into the desired direction.
Scenario	1. Server gets movement data from keyboard or joystick 2. Server will send Sentinel related movement command
Exceptional Situations & Alternative Flows	Connection between the Remote Computer System and Sentinel might be lost, Sentinel stops until the connection is re-established

Use Case Numbers	SRS/4.3, SRS/4.4
Use Case Name	Create 2D Map
Related Use Cases	Publish Data to ROS Node,Get Data From ROS Node
Description	Creates 2D map using sensor data
Inputs	Sensor data
Source	Sentinel
Precondition	Server must be on and connected to the working car. Sensors must be working.
Postcondition	Accurate 2D map must be created.
Scenario	1. The remote server will process sensor data. 2. Then, the remote server will create the map using the data with built-in library RTAB-Map
Exceptional Situations & Alternative Flows	-

Use Case Numbers	SRS/4.3, SRS/4.4, SRS/4.5
Use Case Name	Create 3D Map
Related Use Cases	Create 2D Map, Publish Data to ROS Node,Get Data From ROS Node
Description	Creates 3D map using sensor data and camera data.
Inputs	Sensor data and camera data
Source	Sentinel
Precondition	Server must be on and connected to the working car. Sensors must be working.
Postcondition	Accurate 3D map must be created.
Scenario	1. Server will process sensor data and camera data 2. Server will create the 3D map using the sensor data and camera data together using RTAB-Map
Exceptional Situations & Alternative Flows	-

Use Case Numbers	SRS/4.3
Use Case Name	Process Data
Related Use Cases	Get Data From ROS Node
Description	Process the collected sensor and camera data, then synchronize and calibrate the camera data with the sensor data.
Inputs	Sensor and camera data
Source	Remote Control System
Precondition	Server, sensors and camera must be available.
Postcondition	The processed data must be valid for creating 2D and 3D maps, as well as for path planning.
Scenario	1. Get collected data from client 2. Synchronize data flow between camera and sensors 3. Calibrate camera data with other sensors
Exceptional Situations & Alternative Flows	The incoming data from sensors and camera might be lost during transmission. The system attempts to retransmit the lost data.

Use Case Numbers	SRS/4.1, SRS/4.4
Use Case Name	Path Planning
Related Use Cases	Process Data, Get Data From ROS Node, Set Movement Command
Description	Plan the path and movement using path planning algorithms based on the collected and calibrated data.
Inputs	Calibrated sensor and camera data
Source	Remote Control System
Precondition	Server and sensors must be available. Camera and sensor data must be processed
Postcondition	Planned path must be reachable by The Sentinel
Scenario	1. Runs path planning algorithms according to processed data 2. Find best paths
Exceptional Situations & Alternative Flows	Generated moves may be invalid, or the planned map might be unreachable. The system recalculates the map and adjusts the moves accordingly.

Use Case Numbers	SRS/4.5
Use Case Name	Process the Data for Object Detection
Related Use Cases	Create 3D Map
Description	Remote Computer System process the data for Object Detection using YOLO library.
Inputs	Camera Data
Source	Remote Computer System
Precondition	Connection between the Remote Computer System and Sentinel must be stable. Also, 3D Map must be created.
Postcondition	Remote Computer System completed the Object detection on 3D Map objects.
Scenario	1. The server processes the camera data for object detection using YOLO. 2. The Remote Computer System detects the object on the 3D Map.
Exceptional Situations & Alternative Flows	The connection between the Remote Computer System and Sentinel might be lost, and If a 3D Map cannot created, Object detection will not happen.

Use Case Numbers	SRS/4.5
Use Case Name	Integrate with 3D Mapping
Related Use Cases	Process the Data for Object Detection, Show the Result
Description	After the completed Object Detection, detection objects must be integrated with the 3D Mapping to visualize better.
Inputs	3D Mapping, Detection Objects
Source	Remote Computer System
Precondition	Processing the Camera Data for Object detection must be completed.
Postcondition	The Remote Computer System completed the integrating of 3D Mapping and Detection of Objects.
Scenario	1. The server processes the 3D Mapping and detection objects together. 2. The Remote Computer System integrated 3D Map and Detection object results.
Exceptional Situations & Alternative Flows	Object detection might not be completed. If the server processes the camera data to detect the objects, this use case must wait.

Use Case Numbers	SRS/4.5
Use Case Name	Show the Result
Related Use Cases	Integrate with 3D Mapping
Description	After the completed the integration of 3D Mapping with object detection, the final result is shown to the user.
Inputs	3D Mapping with object detection
Source	Remote Computer System
Precondition	Integration of 3D Mapping with object detection must be completed.
Postcondition	The Remote Computer System shows the results for the user at the remote computer.
Scenario	1. The server completes the integration between 3D mapping and detection objects. 2. After the completion, show the results to the user.
Exceptional Situations & Alternative Flows	The server cannot complete the integration because of the lack of detection objects or 3D mapping creating errors.

Use Case Numbers	SRS/4.5
Use Case Name	Choose the 3D Map with Object Detection mode
Related Use Cases	-
Description	Users at the Remote Computer System, choose the 3D Map with object detection mode to see the Map with the detection objects in the area.
Inputs	User action.
Source	User at the Remote Computer System
Precondition	The user should connect to the server to see the mode options.
Postcondition	Users can see the 3D mapping with object detection.
Scenario	1. The user start the using system at the remote computer 2. Users choose the 3D Mapping with object detection mode. 3. Users can see the 3D Mapping with detection objects
Exceptional Situations & Alternative Flow	The server might not have completed integration between 3D Mapping and detection objects. Therefore user cannot see the results.

3.3.3. Non-Functional Requirements

3.3.3.1. Performance Requirements

Performance Requirements	Description
Response Time	While real-time video is displayed and path planning algorithms are running, response time must be lower than 1 second. The system must work nearly real-time.
Error Handling	If an unknown error occurs, the system should restart itself, or the car must be returned to the starting location.
Workload	The system must handle multiple subsystems that are running at the same time. Path planning, video streaming, 2D and 3D mapping, and object classification will be done at the same time.
Scalability	Our vehicle will be controlled by a remote computer. Only one server can work on a vehicle at the same time. System shouldn't crash for any scalability issues.
Application Requirements	The remote server should have 1 GB ram available and a 4 GB graphics card to process images. The remote server CPU must be able to perform multi-processing.

3.3.3.2. Safety Requirements

Safety Requirements	**Description**
Damage Preventing	The vehicle must make quick and effective maneuvers to avoid obstacles and harmful objects to avoid getting or doing damage.
Error Reporting	Every system report must be reported in under 2 seconds. Every crucial report must be reported in under 1 second.
Switchable Modes	The system must allow its user to switch modes between autonomous and manual driving, so that the user can take over the errors.
Obstacle Detection	The system must detect obstacles with high accuracy.

3.3.3.3. Security Requirements

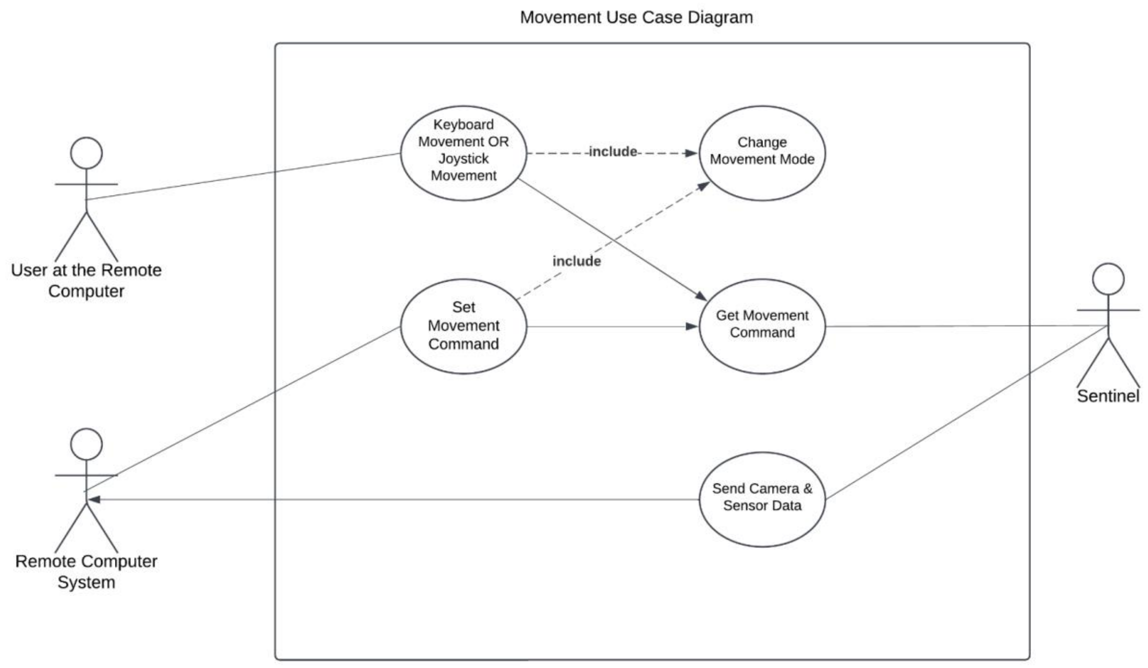
Security Requirements	**Description**
Wifi Access	The connected wifi must be encrypted with a secure password and the must word must be hidden from the public.
Server Setting & Keys	The server settings and access keys must be protected.

3.3.3.4. Software Quality Attributes

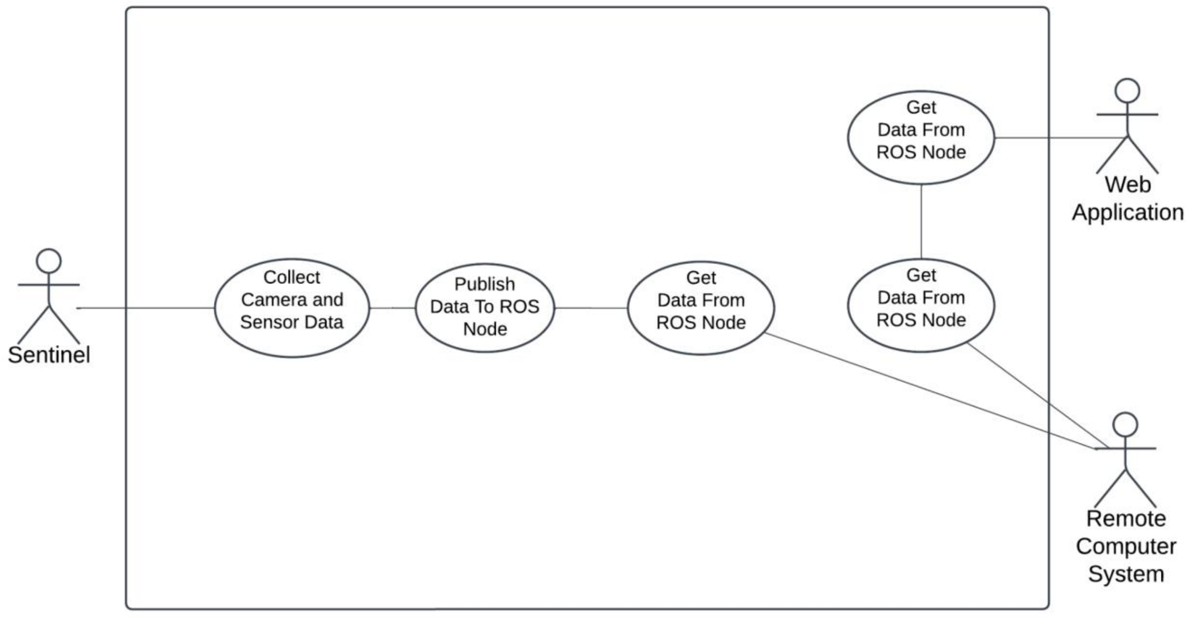
Software Quality Attributes	**Description**
Reliability	The system shouldn’t produce errors in normal conditions and complete the task successfully.
Robustness	Under heavy environmental conditions the vehicle must be able to complete its tasks.
Portability	The system should work on Ubuntu and the server should work on Windows.
Correctness	Accuracy of predicting obstacles, classifying objects and creating maps must be over 85%.
Learnability	The usage of the system should be easy to understand for a strange user. Users must understand the concept in under 2 hours.
Maintainability	After critical error occurs, the system must restart itself without any loss of information.
Testability	The system must be tested on the different areas and different environmental conditions.
Efficiency	The system should use its resources effectively.
Usability	The vehicle should be controlled by a joystick or keyboard from a remote server.
Autonomy	The car must complete its tasks without human interruption.
Modifiability	The system may be updated, or new features will be added in the future. The system design must allow these changes.

3.4. Use Cases

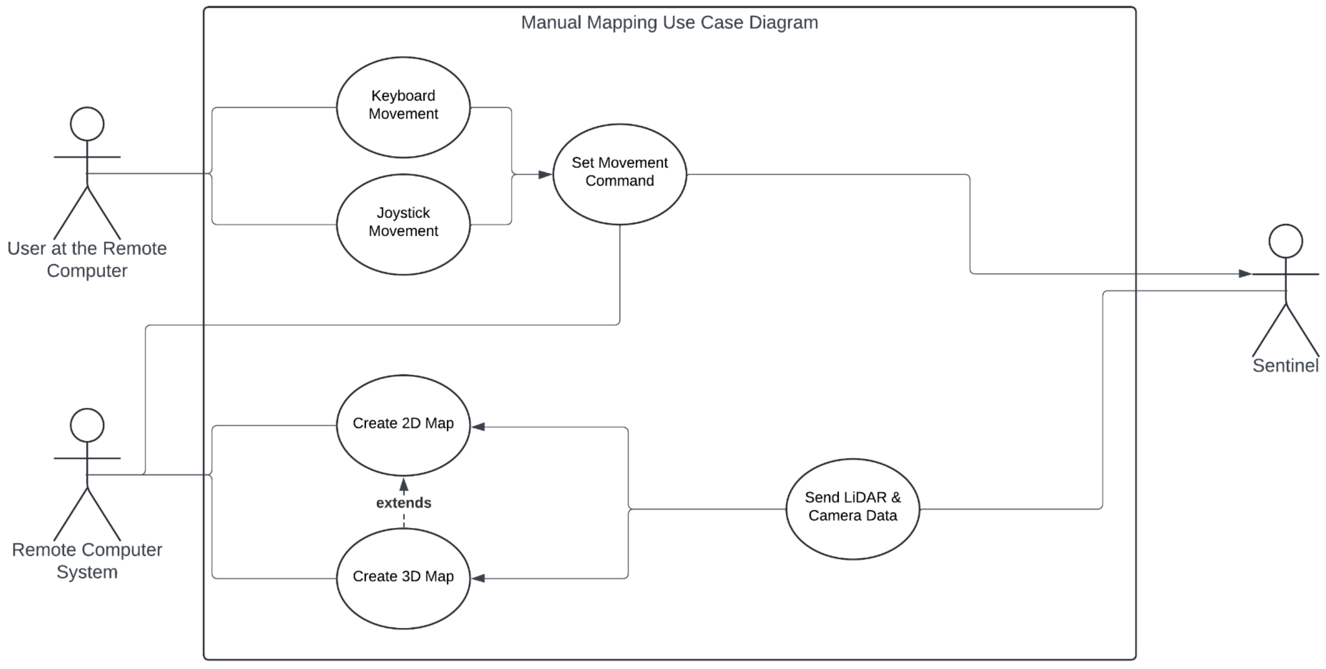
3.4.1. Movement


Movement Use Case Diagram
Keyboard Movement OR Joystick Movement
include
Change Movement Mode
include
Set Movement Command
Get Movement Command
Send Camera & Sensor Data
User at the Remote Computer
Remote Computer System
Sentinel

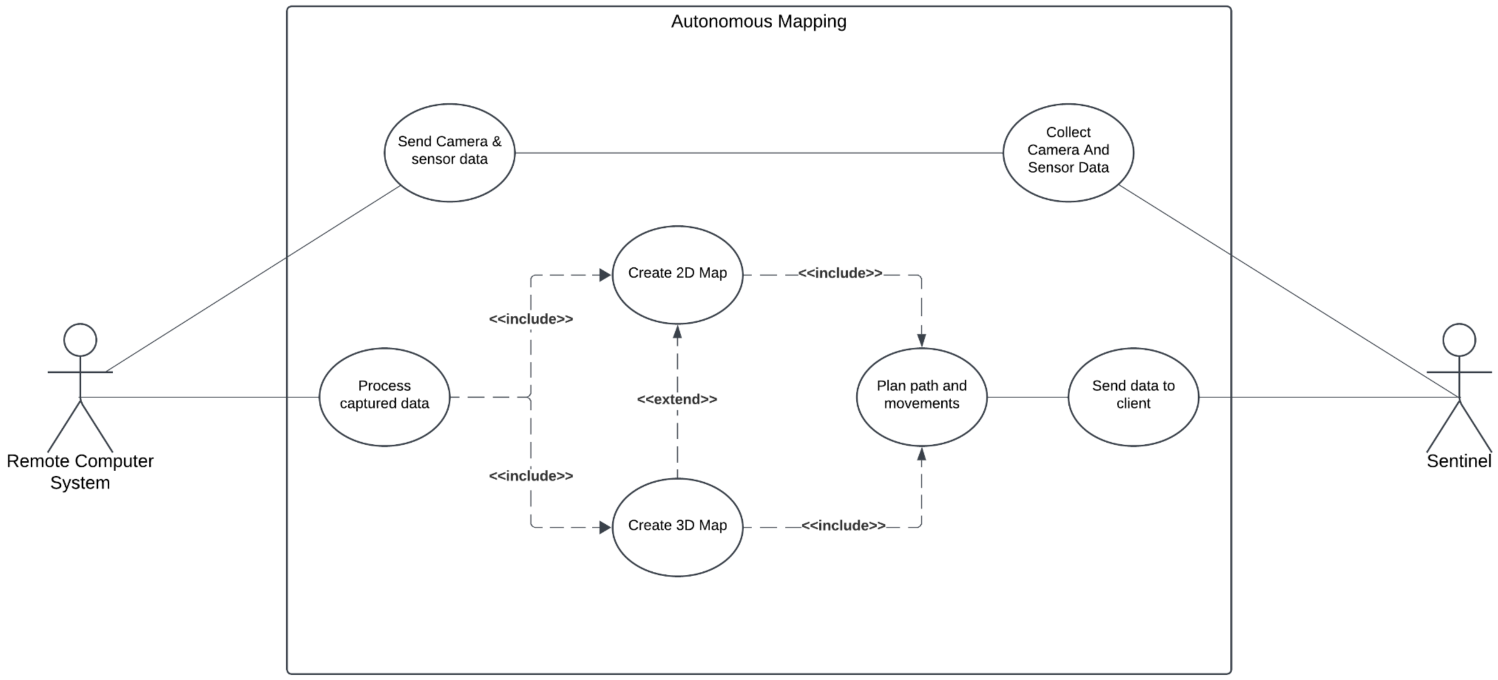
3.4.2. Data Stream



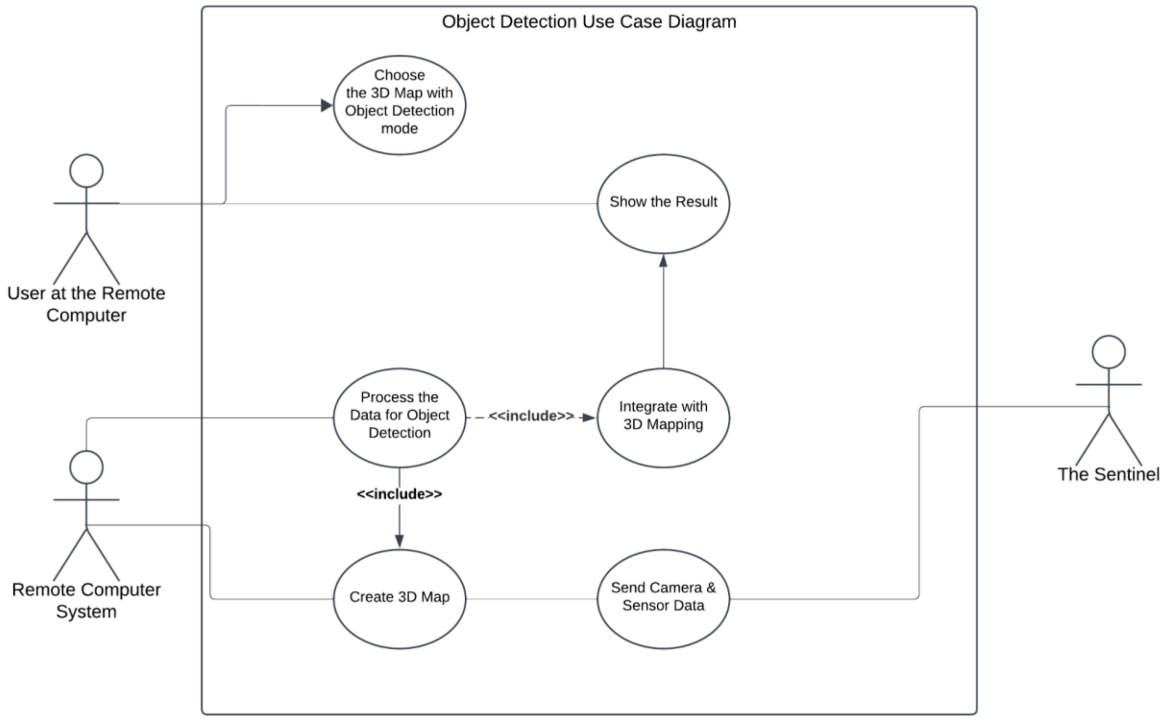
3.4.3. Manual Mapping



3.4.4. Autonomous Mapping



3.4.5. Object Detection


Object Detection Use Case Diagram
Choose the 3D Map with Object Detection mode
Show the Result
User at the Remote Computer
Process the Data for Object Detection
<<include>>
Integrate with 3D Mapping
<<include>>
Create 3D Map
Send Camera & Sensor Data
Remote Computer System
The Sentinel

4. Software Design Document

4.1. Introduction

4.1.1. Purpose of this Document

The purpose of this document is to provide a detailed presentation of the structure and architecture of the software included in Sentinel. The document aims to be reference material for developers, stakeholders, and team members by making the design decisions and implementation techniques in the operation of the system clearer. The document mainly includes sequence diagrams, activity diagrams, data flow diagrams (DFD), and user interface designs that help users and team members to get a better understanding of the system's functions, workflows, and interactions. This approach not only aims to facilitate effective project management through clear communication among team members, but also plays an important role in being a guide for future changes or iterations.

4.1.2. Scope of this Document

The Sentinel idea stemmed from the development of unmanned vehicle technologies in modern warfare. In today's world, many countries use technology in the defense industry for both self-defense and operational purposes. When these vehicles are not used, they cause many soldiers to lose their lives. For this reason, there is a great need for unmanned vehicles in war zones. For this reason, many countries are investing in these vehicles. This situation caught our interest and we decided to make an autonomous environment mapping vehicle. However, our vehicle is a proof of concept and isn't suitable for the war zone. The main purpose of Sentinel is to create 2D and 3D maps of its surroundings by moving autonomously. The vehicle can be sent to dangerous places to autonomously map the environment, thus preventing personnel loss. It can be used to create a map of the environment in many areas such as buildings that are not safe to enter, war zones and places contaminated with harmful gases. The importance of this is that it gives an idea about the place without human interaction and provides detailed 3D environment mapping that shows dangers in advance. It can also be used to detect life in rescue operations when a thermal camera is used. The vehicle is mainly equipped with Raspberry Pi5, which runs on Ubuntu, YDLidar X2, and Picamera 3. The data collected from the lidar sensor and camera are combined and used to generate the 3D map utilizing the rtabmap_ros, a package used to generate a 3D point of clouds of the environment and/or to create a 2D occupancy grid map for navigation. It will mainly use the Robot Operating System (ROS) to control both autonomous and manual movement, and environment mapping. The collected data is planned to be processed at a remote computer that is also equipped with ROS, and only the movement command from the remote computer will be returned to Sentinel. In this way, we also plan to provide a manual control mechanism which allows our users to interact with the vehicle any given time. Moreover, the generated map and the camera footage will be available to the user at the remote computer in real-time. After the map has been generated, we plan to assign jobs to the Sentinel, such as finding the red chair from the generated map, and it will find that red chair. To achieve this, we also plan to detect the objects using YOLO, a library for object detection, and then store these information using rosbag. In summary, users can control the movement of the vehicle, view images from the vehicle, and view 2D and 3D maps in real time. All of this is done on a remote computer. On the other hand, Sentinel is responsible for sending sensor and camera data to the remote computer and obeying the command received from the remote computer. In the following sections of this SDD document, sequence diagrams, activity diagrams, and user interfaces of DFD (Data Flow Diagrams) will be shown to provide the development team with a better understanding and impression of the system functions and interfaces, and also to facilitate future improvements.

4.1.3. Glossary

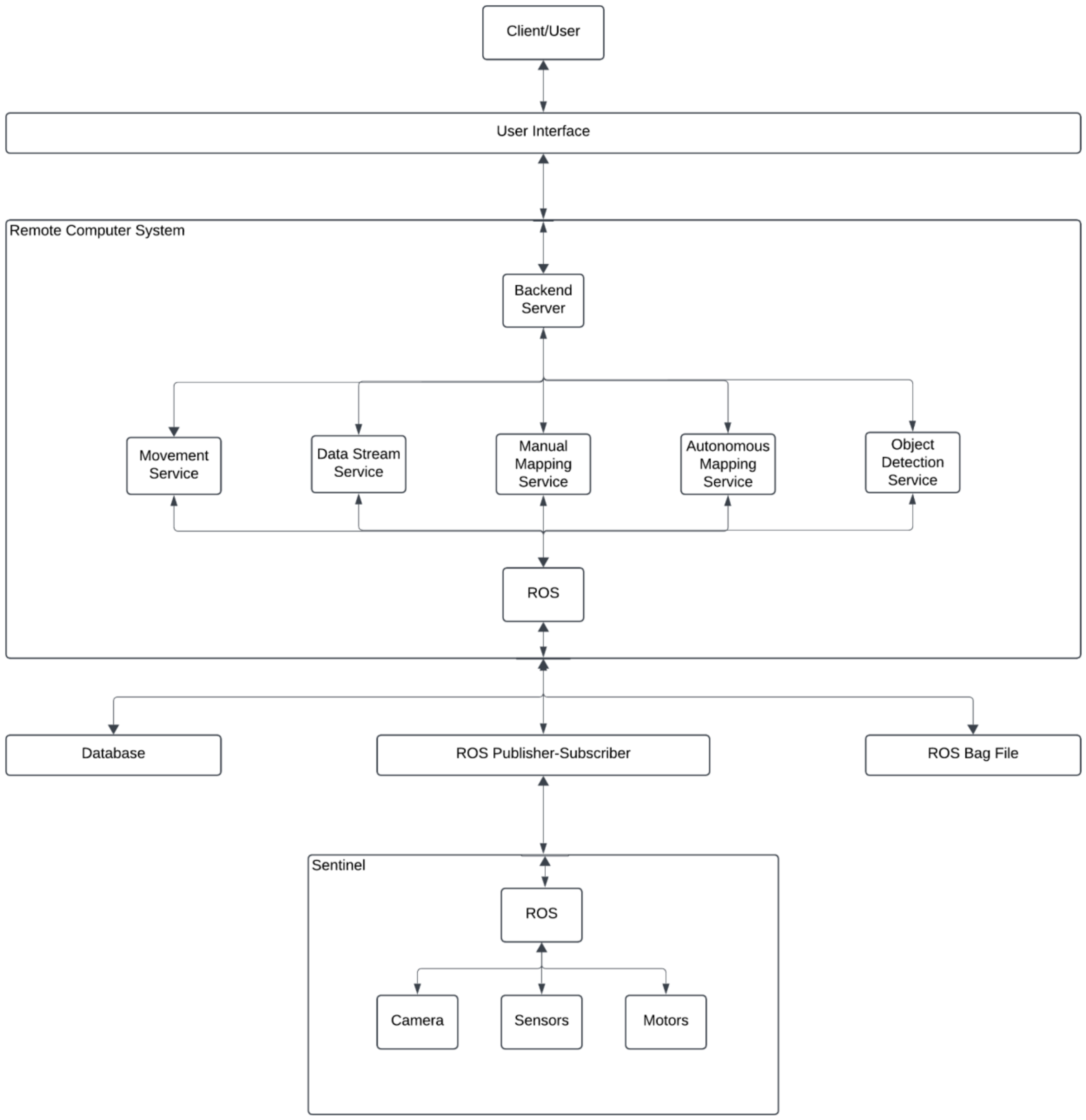
Term	**Description**
Pi Camera Module 3	Compact camera from Raspberry Pi
Raspberry Pi 5	Small single-board computer
ROS Publisher/Subscriber	Publish/subscribe is a messaging pattern where publishers categorize messages into classes that are received by subscribers.
Rosbag	A set of tools for recording and playing back to ROS topics.
RTAB-Map	ROS framework for 3D and 2D mapping.
The Sentinel	The sentinel is a Discovery Vehicle. The name of our project, The Sentinel, is an autonomous vehicle with features such as 3D mapping and object detection.
Ubuntu	Linux based operating system
YDLidar X2	360-degree two-dimensional rangefinder sensor.
YOLO	You Look Only Once. Open source library that helps you object detection.

4.2. Overview of Software Design Document

This report provides an explanation of how the system's designed and functions, and user interfaces. At the beginning, the Architectural Design is discussed to illustrate how the different parts of the system work together. Then, The Data Flow Diagram section presents how information moves within the system starting from the Context Diagram and progressing to more detailed level 1 DFD illustrations, like information movement, data exchange procedures manual and autonomous mapping and object recognition. Class Diagram shows how the system will be implemented and which inheritions will be used. The Activity Diagrams illustrate workflows for tasks to be followed step by step. Sequence Diagram representations give examples for the order of interactions between various system components; ultimately the document encompasses User Interfaces prototypes that help users to understand how layout works. Also, some interaction examples are given in this section.

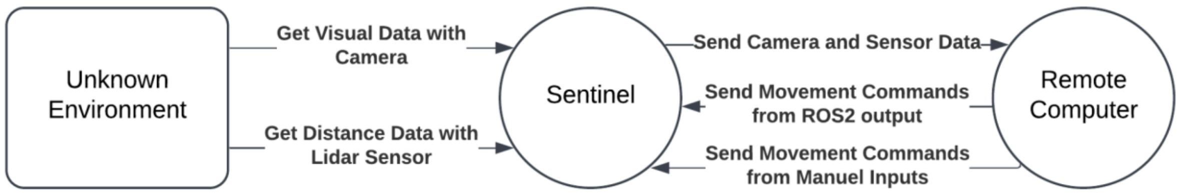
4.3. System Design

4.3.1. Architectural Design



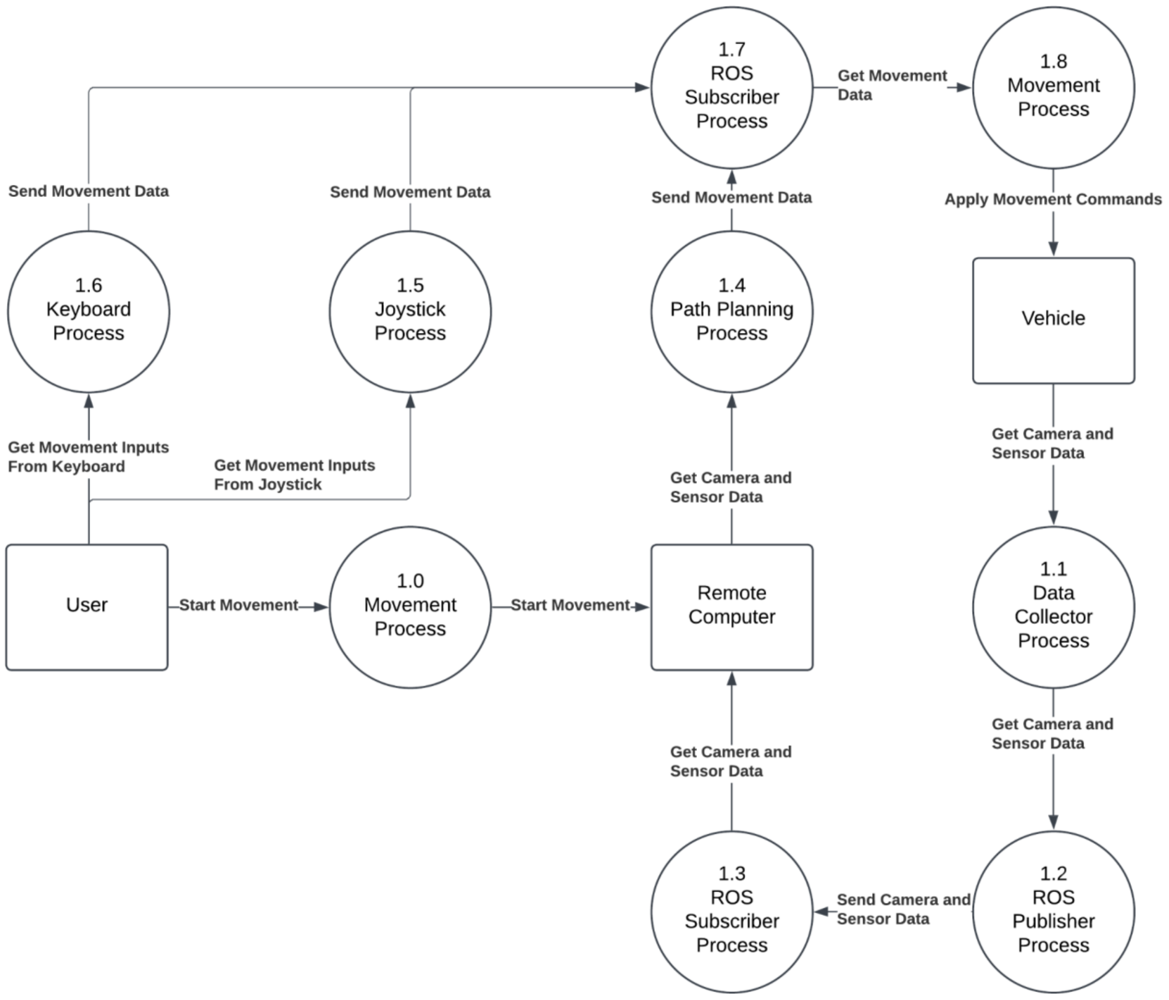
4.3.2. Data Flow Diagrams

4.3.2.1. Context Diagram

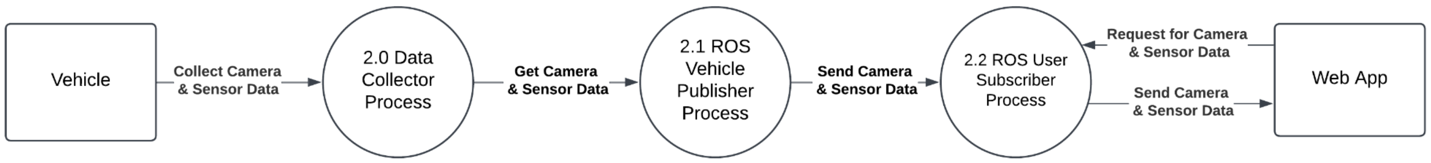

Unknown Environment
Get Visual Data with Camera
Get Distance Data with Lidar Sensor
Sentinel
Send Camera and Sensor Data
Send Movement Commands from ROS2 output
Send Movement Commands from Manuel Inputs
Remote Computer

4.3.2.2. Level 1 DFDs

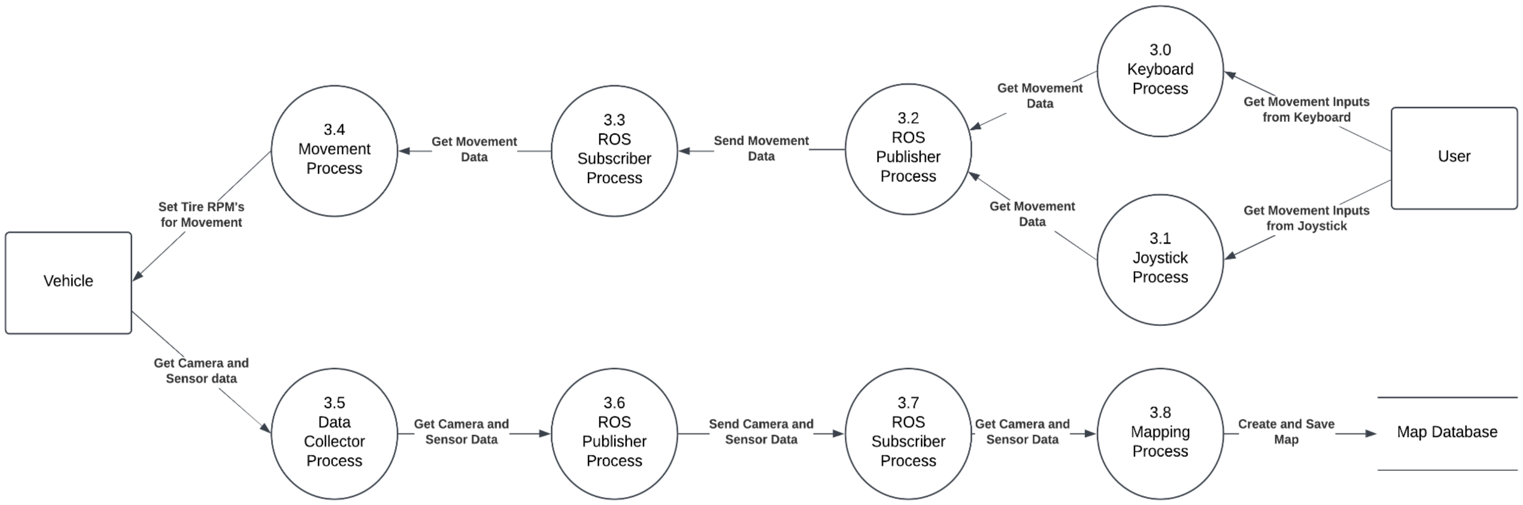
4.3.2.2.1. Movement


1.7 ROS Subscriber Process
Get Movement Data
1.8 Movement Process
Send Movement Data
Send Movement Data
Send Movement Data
Apply Movement Commands
1.6 Keyboard Process
1.5 Joystick Process
1.4 Path Planning Process
Vehicle
Get Movement Inputs From Keyboard
Get Movement Inputs From Joystick
Get Camera and Sensor Data
Get Camera and Sensor Data
User
Start Movement
1.0 Movement Process
Start Movement
Remote Computer
1.1 Data Collector Process
Get Camera and Sensor Data
Get Camera and Sensor Data
1.3 ROS Subscriber Process
Send Camera and Sensor Data
1.2 ROS Publisher Process

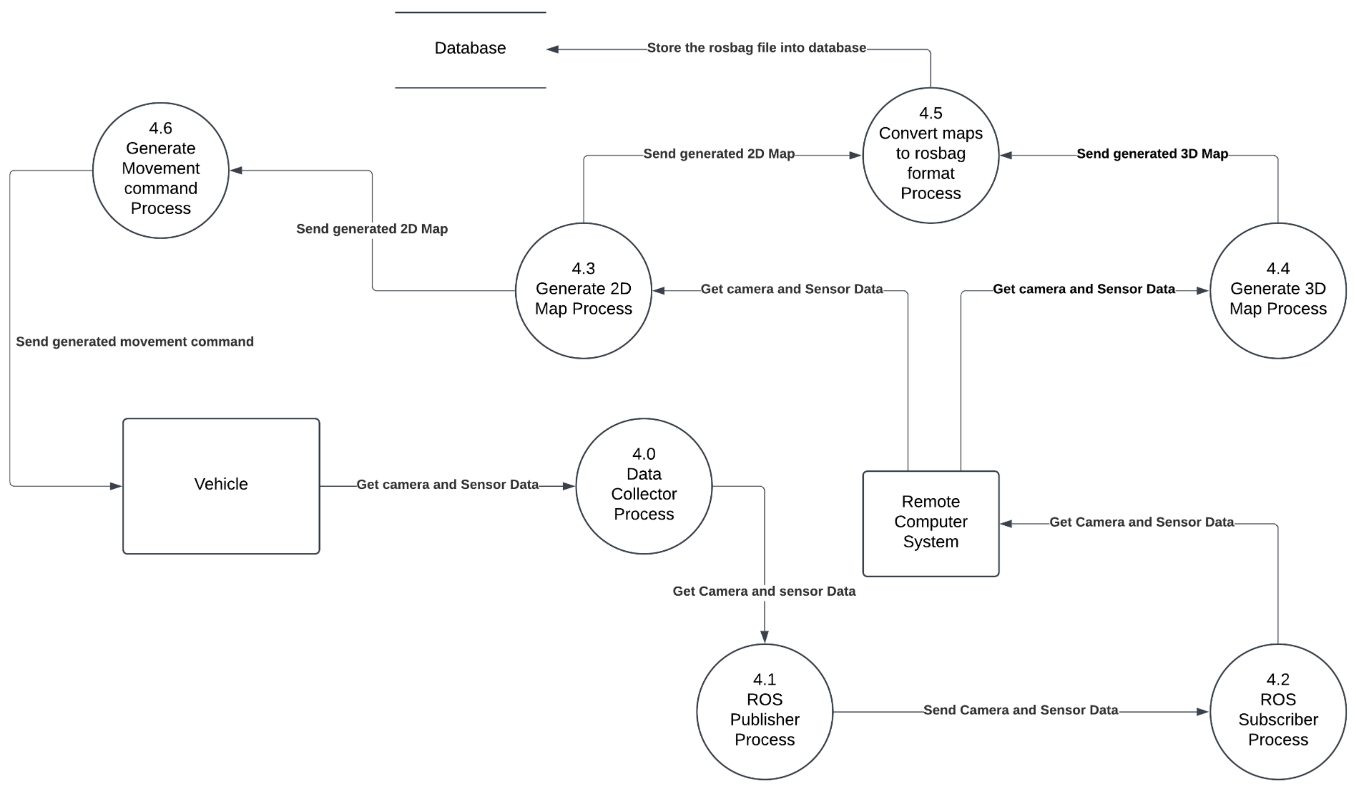
4.3.2.2.2. Data Stream


Vehicle
Collect Camera & Sensor Data
2.0 Data Collector Process
Get Camera & Sensor Data
2.1 ROS Vehicle Publisher Process
Send Camera & Sensor Data
2.2 ROS User Subscriber Process
Request for Camera & Sensor Data
Send Camera & Sensor Data
Web App

4.3.2.2.3. Manual Mapping

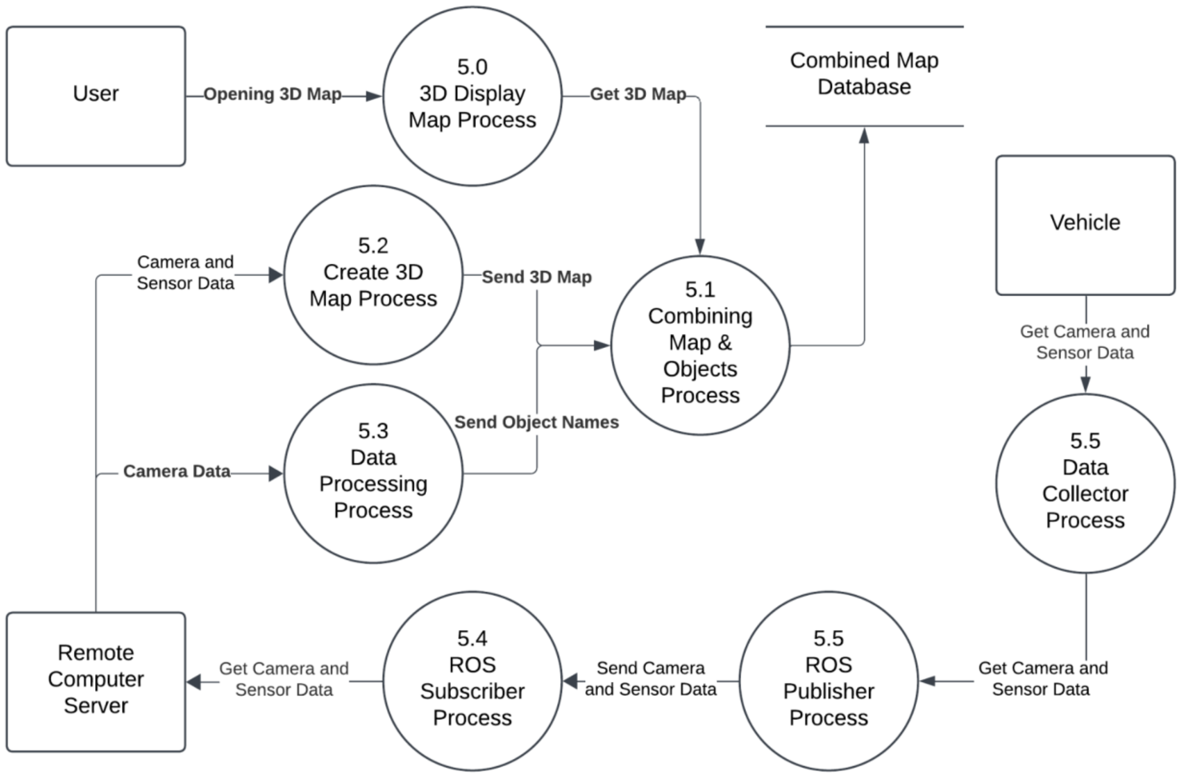

3.0 Keyboard Process
3.1 Joystick Process
User
Get Movement Inputs from Keyboard
Get Movement Inputs from Joystick
Get Movement Data
Get Movement Data
3.2 ROS Publisher Process
Send Movement Data
3.3 ROS Subscriber Process
Get Movement Data
3.4 Movement Process
Set Tire RPM's for Movement
Vehicle
Get Camera and Sensor data
3.5 Data Collector Process
Get Camera and Sensor Data
3.6 ROS Publisher Process
Send Camera and Sensor Data
3.7 ROS Subscriber Process
Get Camera and Sensor Data
3.8 Mapping Process
Create and Save Map
Map Database

4.3.2.2.4. Autonomous Mapping

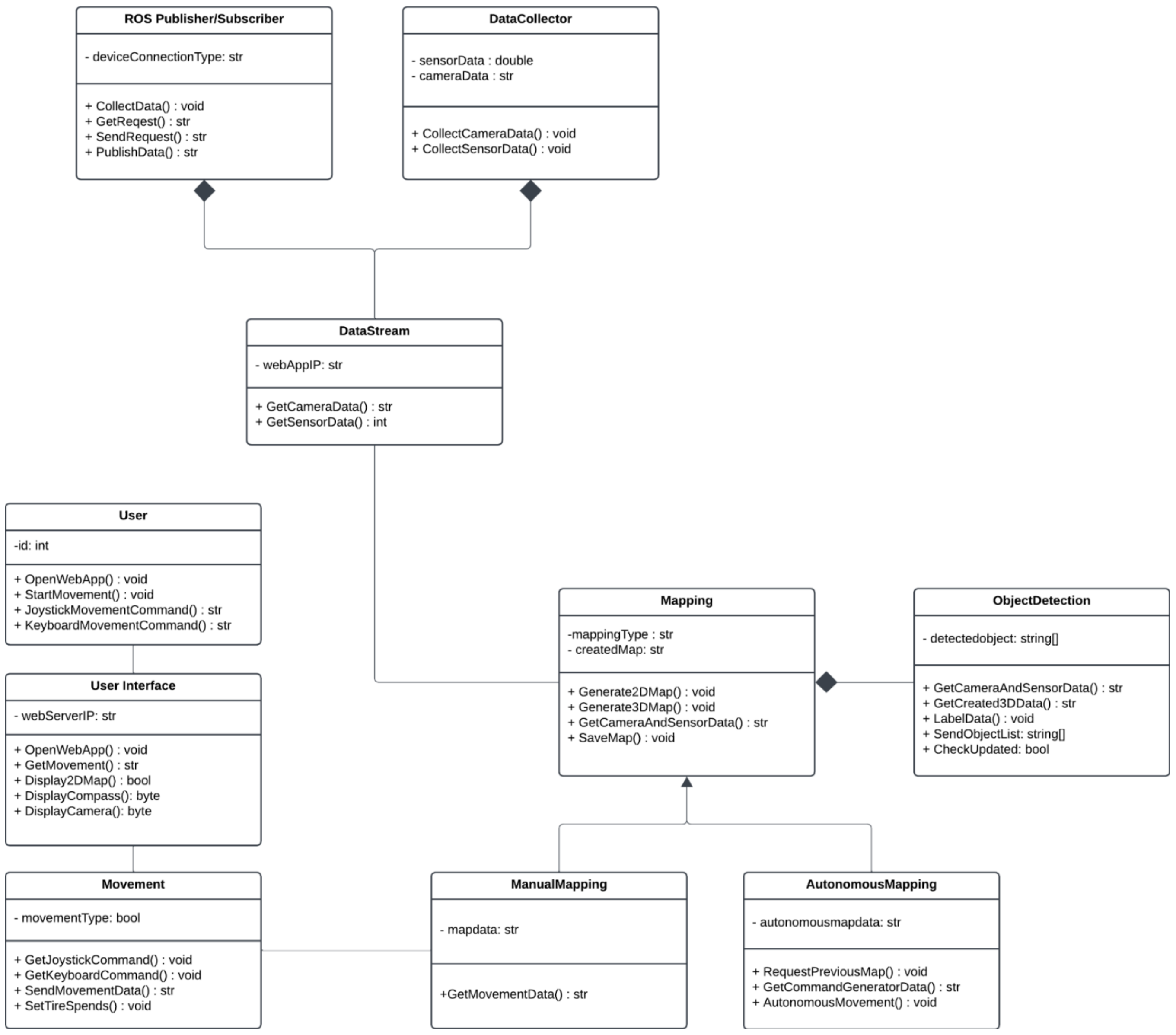


Database
Store the rosbag file into database
4.5 Convert maps to rosbag format Process
Send generated 2D Map
Send generated 3D Map
4.6 Generate Movement command Process
Send generated 2D Map
4.3 Generate 2D Map Process
Get camera and Sensor Data
Get camera and Sensor Data
4.4 Generate 3D Map Process
Send generated movement command
Vehicle
Get camera and Sensor Data
4.0 Data Collector Process
Remote Computer System
Get Camera and Sensor Data
Get Camera and sensor Data
4.1 ROS Publisher Process
Send Camera and Sensor Data
4.2 ROS Subscriber Process

4.3.2.2.5. Object Detection

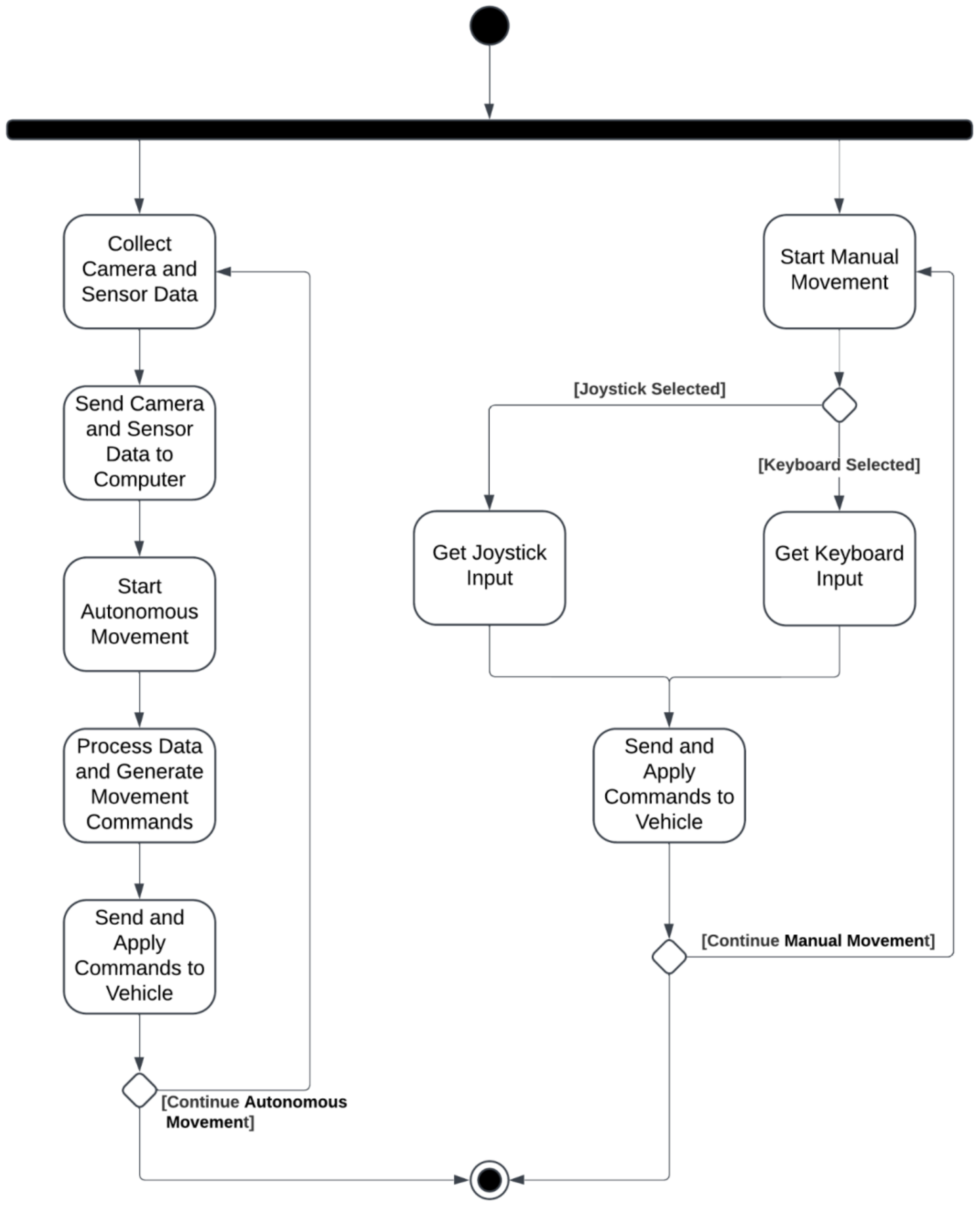

User
Opening 3D Map
5.0 3D Display Map Process
Get 3D Map
Combined Map Database
Camera and Sensor Data
5.2 Create 3D Map Process
Send 3D Map
5.1 Combining Map & Objects Process
Vehicle
Get Camera and Sensor Data
Send Object Names
5.3 Data Processing Process
Camera Data
5.5 Data Collector Process
Remote Computer Server
Get Camera and Sensor Data
5.4 ROS Subscriber Process
Send Camera and Sensor Data
5.5 ROS Publisher Process
Get Camera and Sensor Data

4.3.3. Class Diagrams



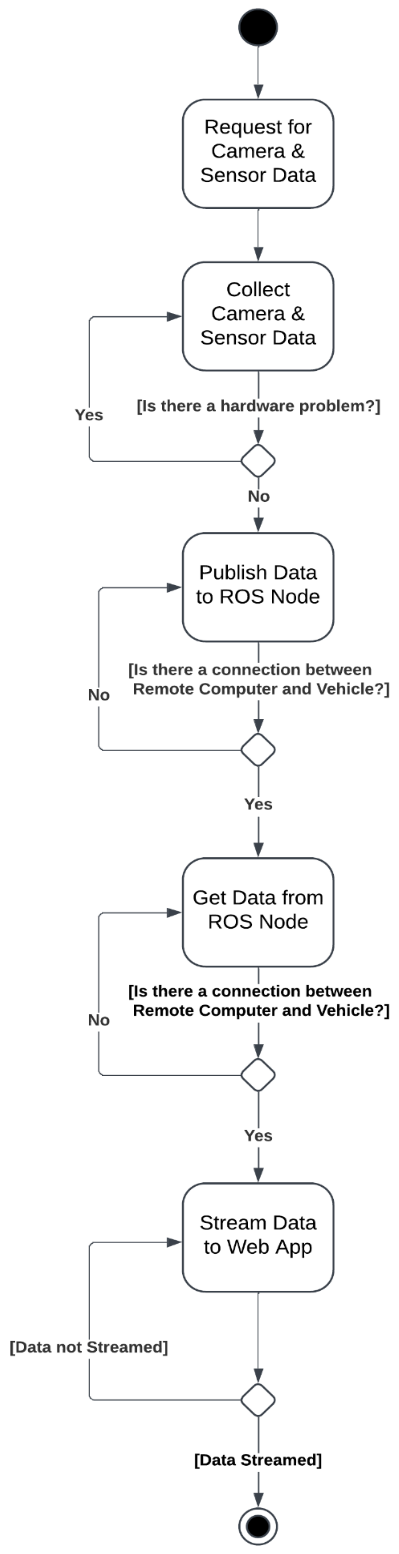
4.3.4. Activity Diagrams

4.3.4.1. Movement

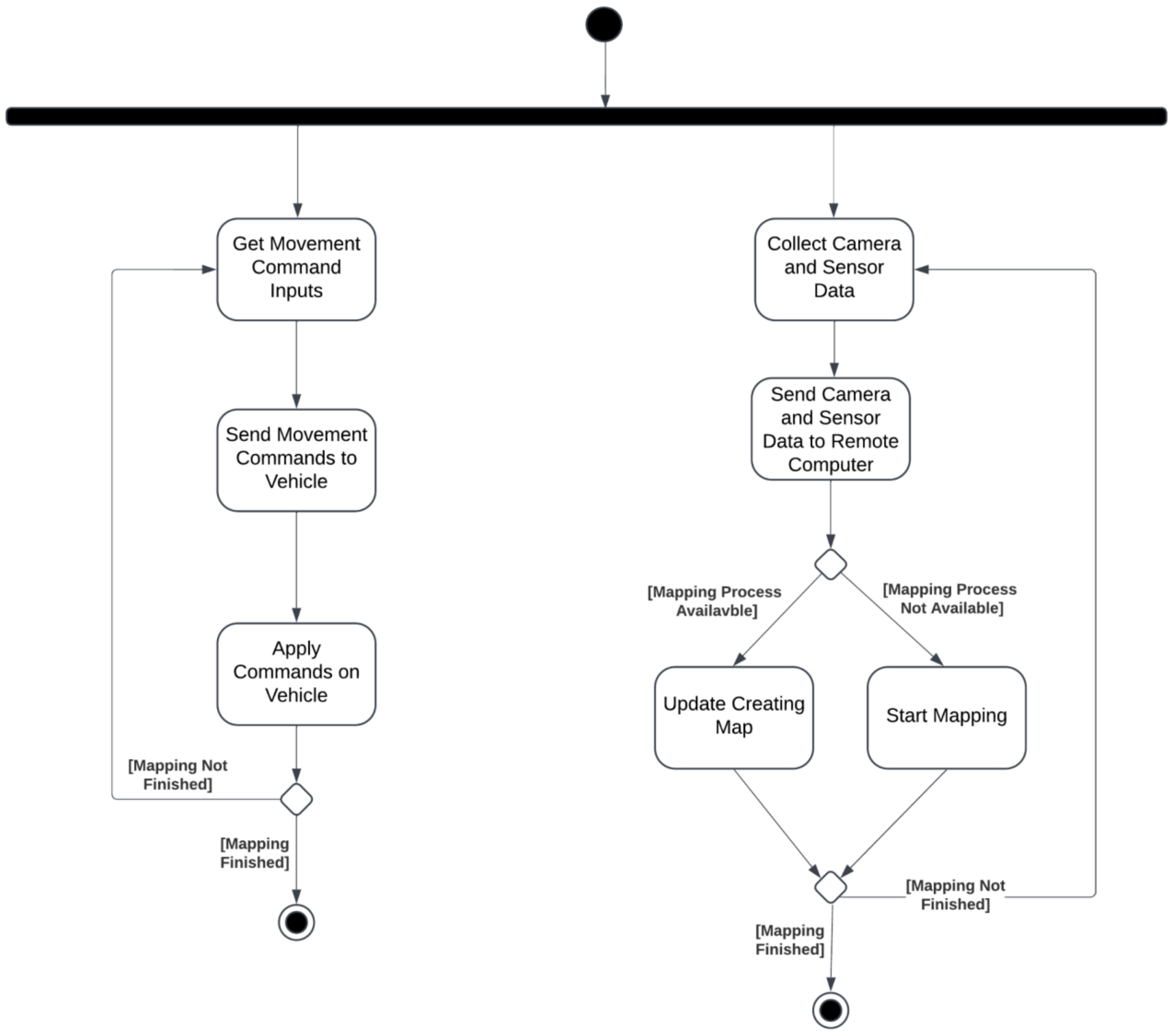


Collect Camera and Sensor Data
Send Camera and Sensor Data to Computer
Start Autonomous Movement
Process Data and Generate Movement Commands
Send and Apply Commands to Vehicle
[Continue Autonomous Movement]
Start Manual Movement
[Joystick Selected]
[Keyboard Selected]
Get Joystick Input
Get Keyboard Input
Send and Apply Commands to Vehicle
[Continue Manual Movement]

4.3.4.2. Data Stream

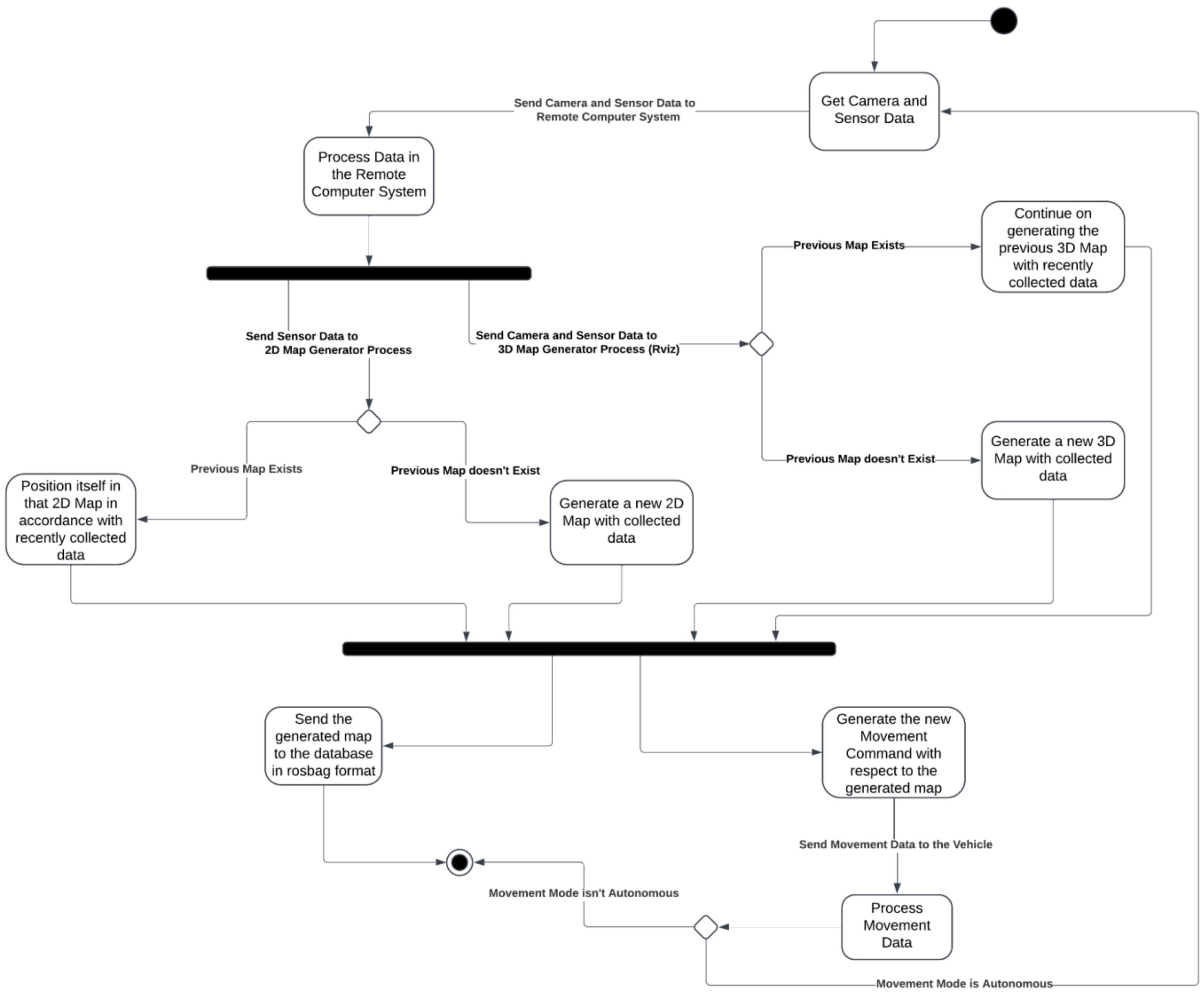

Request for Camera & Sensor Data
Collect Camera & Sensor Data
[Is there a hardware problem?]
Yes
No
Publish Data to ROS Node
[Is there a connection between Remote Computer and Vehicle?]
No
Yes
Get Data from ROS Node
[Is there a connection between Remote Computer and Vehicle?]
No
Yes
Stream Data to Web App
[Data not Streamed]
[Data Streamed]

4.3.4.3. Manual Mapping

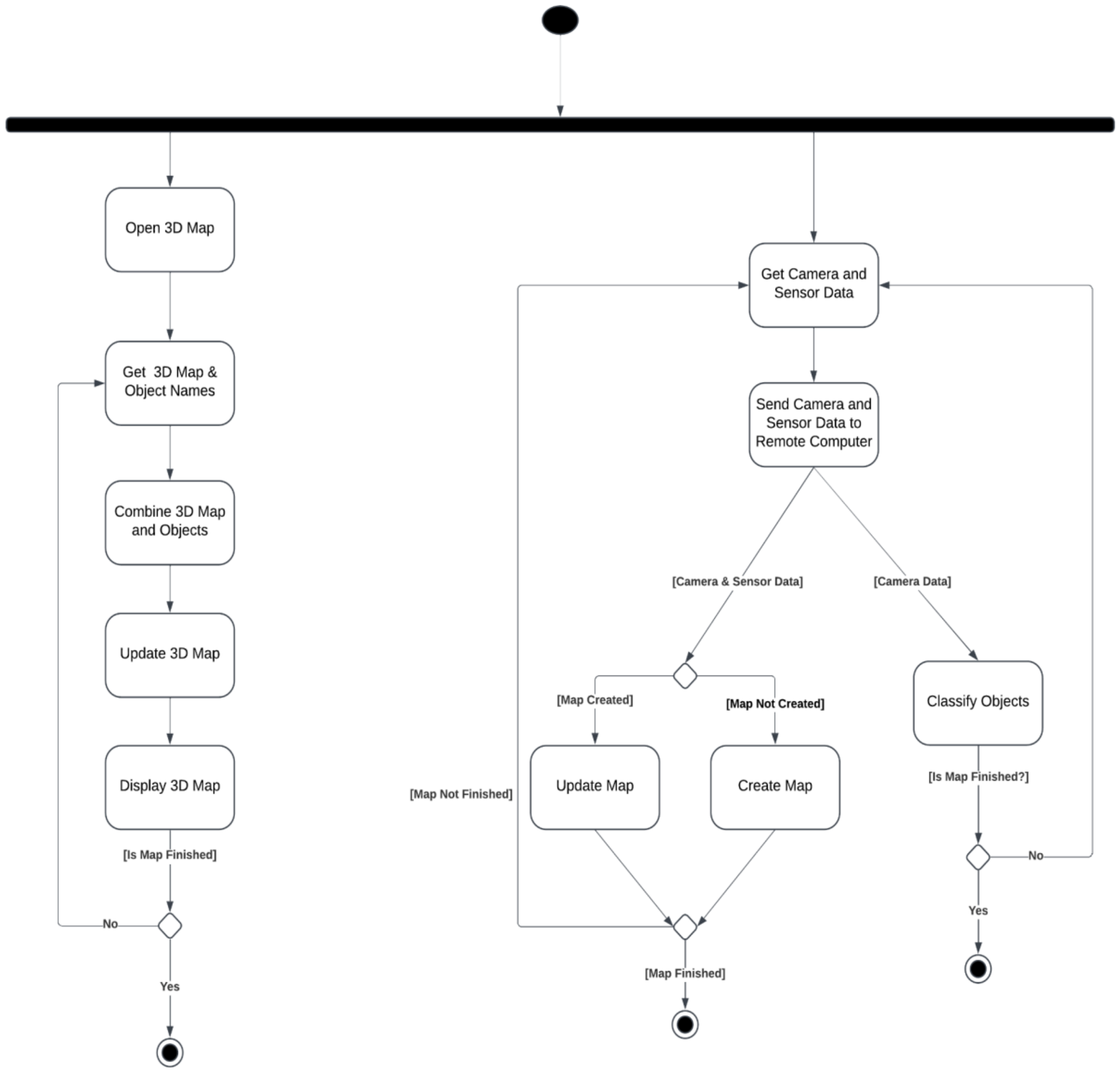


Get Movement Command Inputs
Send Movement Commands to Vehicle
Apply Commands on Vehicle
[Mapping Not Finished]
[Mapping Finished]
Collect Camera and Sensor Data
Send Camera and Sensor Data to Remote Computer
[Mapping Process Availavble]
[Mapping Process Not Available]
Update Creating Map
Start Mapping
[Mapping Not Finished]
[Mapping Finished]

4.3.4.4. Autonomous Mapping

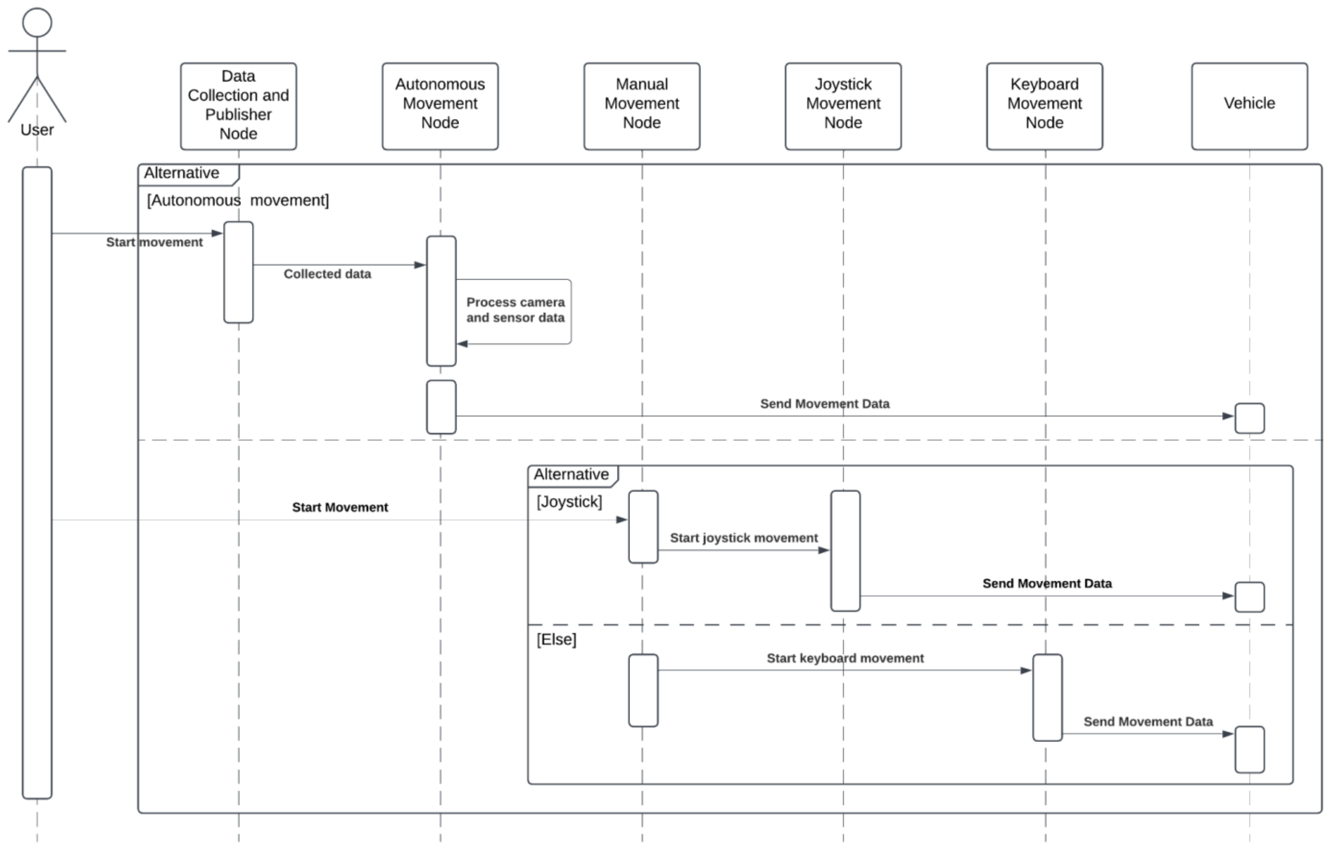

Get Camera and Sensor Data
Send Camera and Sensor Data to Remote Computer System
Process Data in the Remote Computer System
Send Sensor Data to 2D Map Generator Process
Send Camera and Sensor Data to 3D Map Generator Process (Rviz)
Previous Map Exists
Continue on generating the previous 3D Map with recently collected data
Previous Map doesn't Exist
Generate a new 3D Map with collected data
Previous Map Exists
Previous Map doesn't Exist
Position itself in that 2D Map in accordance with recently collected data
Generate a new 2D Map with collected data
Send the generated map to the database in rosbag format
Generate the new Movement Command with respect to the generated map
Send Movement Data to the Vehicle
Process Movement Data
Movement Mode isn't Autonomous
Movement Mode is Autonomous

4.3.4.5. Object Detection

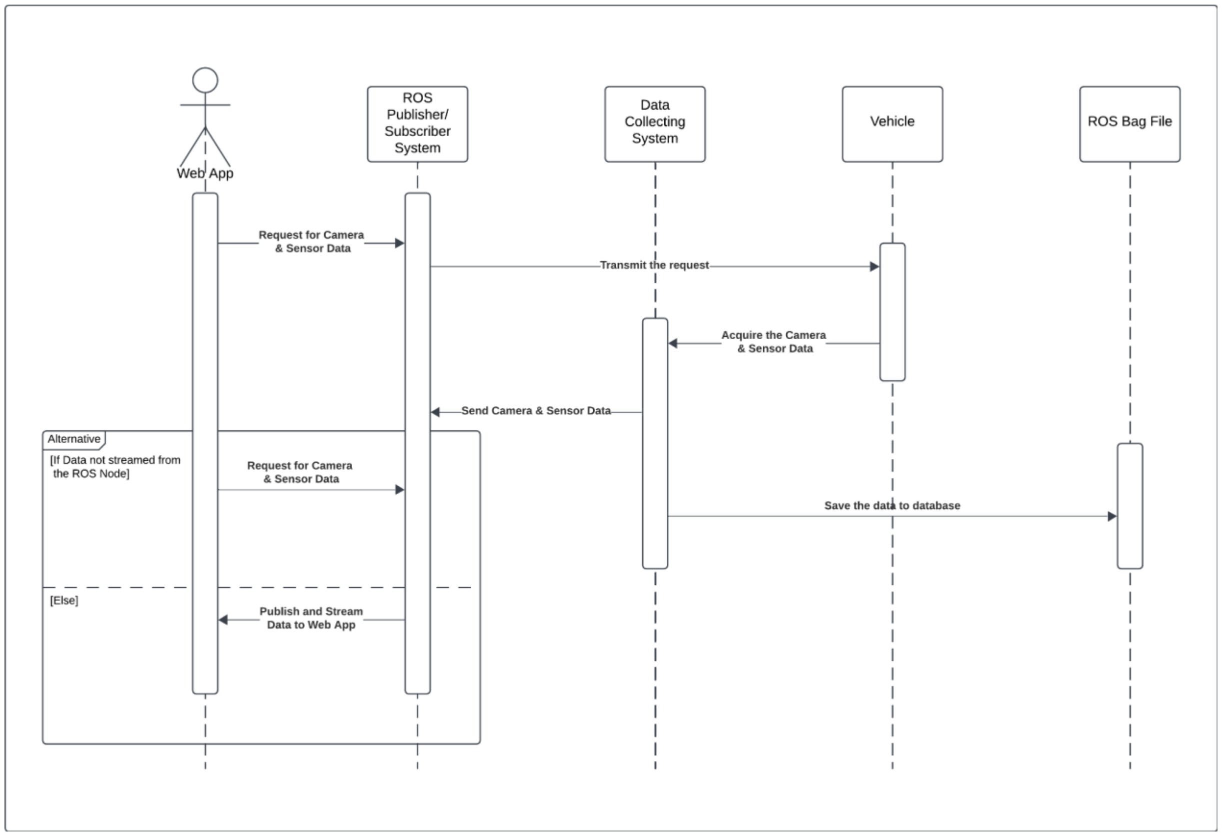


4.3.5. Sequence Diagrams

4.3.5.1. Movement

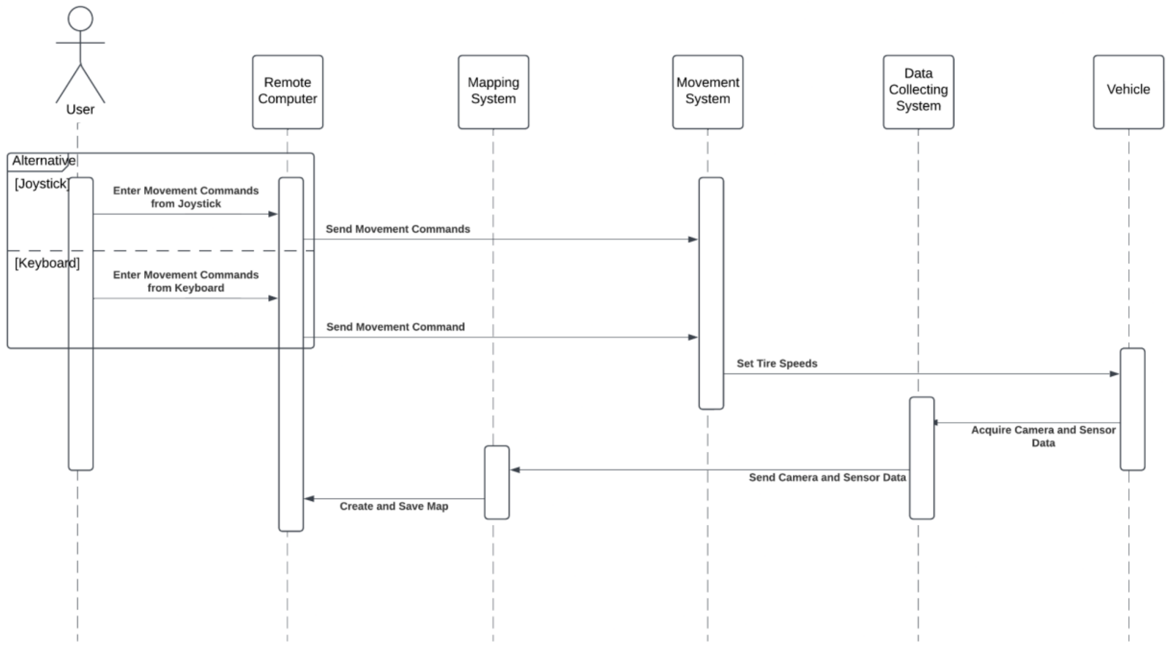


4.3.5.2. Data Stream

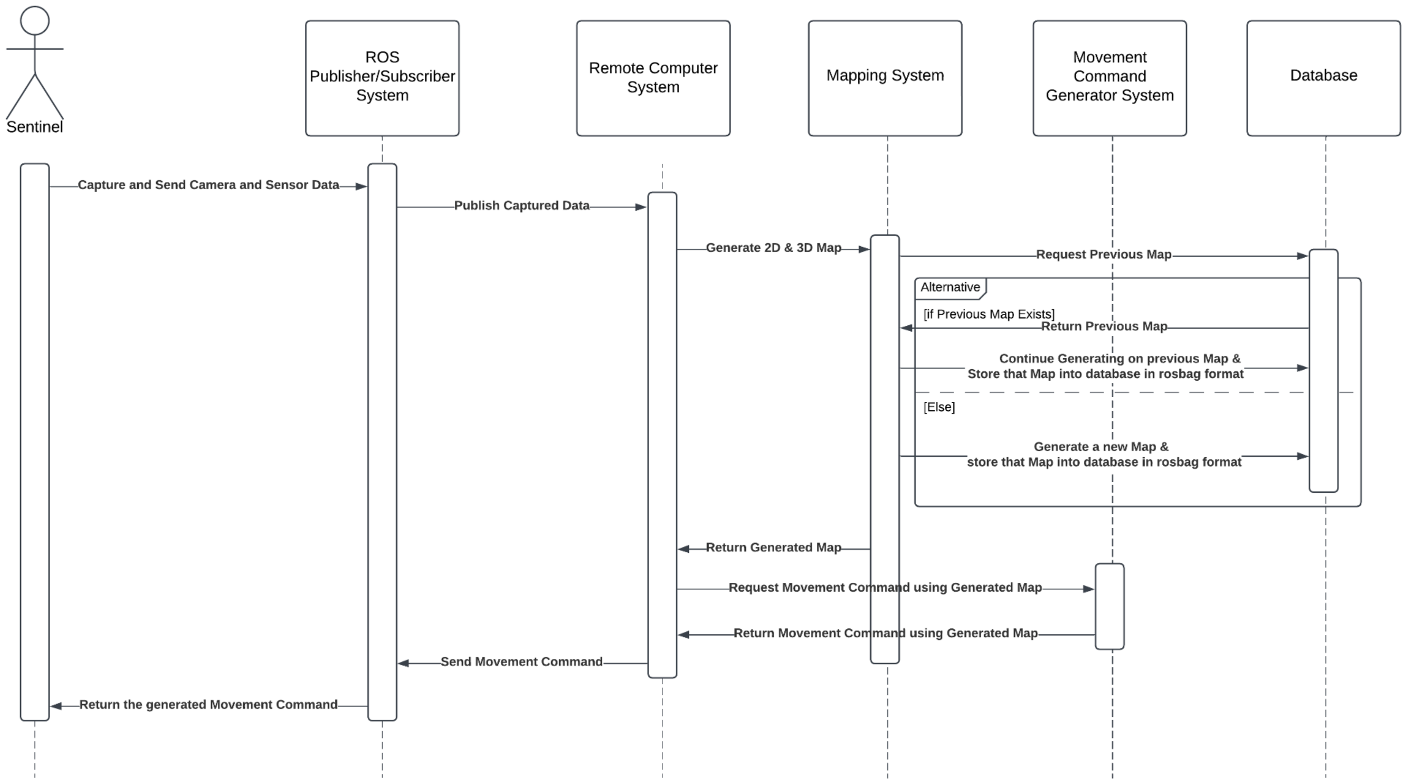


Web App
ROS Publisher/ Subscriber System
Data Collecting System
Vehicle
ROS Bag File
Request for Camera & Sensor Data
Transmit the request
Acquire the Camera & Sensor Data
Send Camera & Sensor Data
Alternative
[If Data not streamed from the ROS Node]
Request for Camera & Sensor Data
Save the data to database
[Else]
Publish and Stream Data to Web App

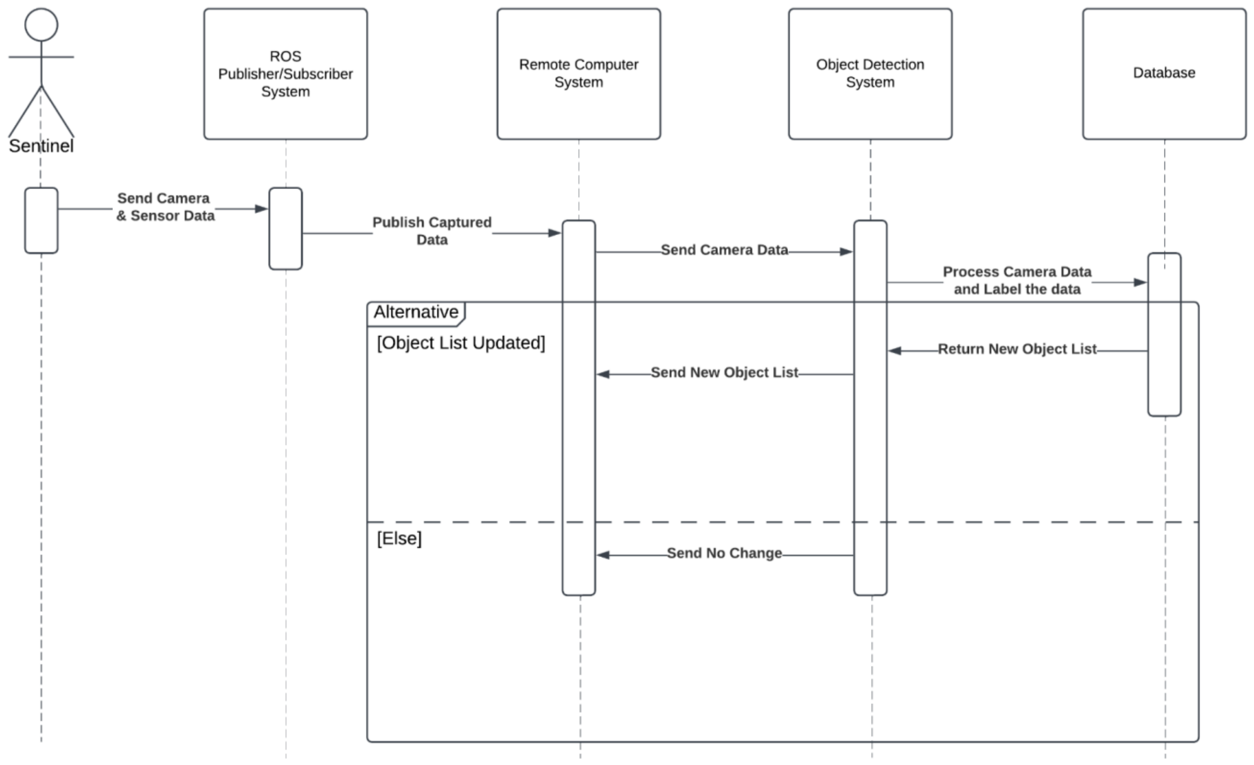
4.3.5.3. Manual Mapping



4.3.5.4. Autonomous Mapping

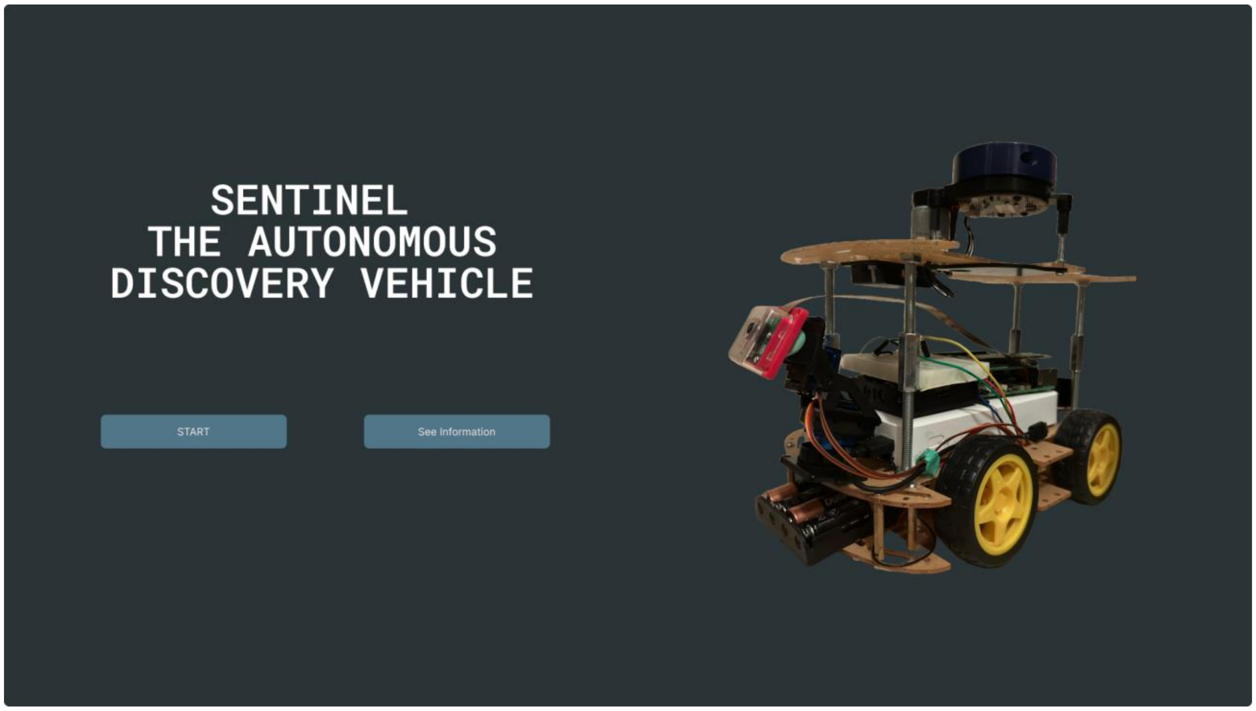

Sentinel
ROS Publisher/Subscriber System
Remote Computer System
Mapping System
Movement Command Generator System
Database
Capture and Send Camera and Sensor Data
Publish Captured Data
Generate 2D & 3D Map
Request Previous Map
Alternative
[if Previous Map Exists]
Return Previous Map
Continue Generating on previous Map & Store that Map into database in rosbag format
[Else]
Generate a new Map & store that Map into database in rosbag format
Return Generated Map
Request Movement Command using Generated Map
Return Movement Command using Generated Map
Send Movement Command
Return the generated Movement Command

4.3.5.5. Object Detection

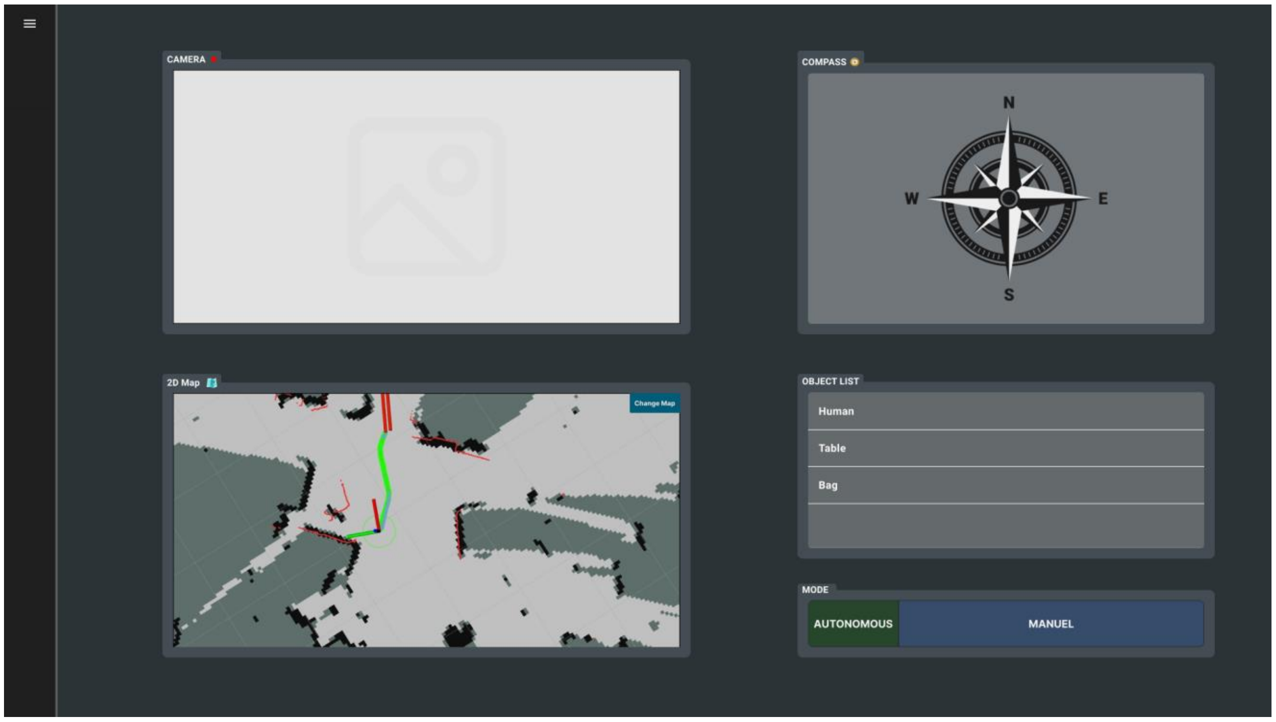


4.4. User Interfaces

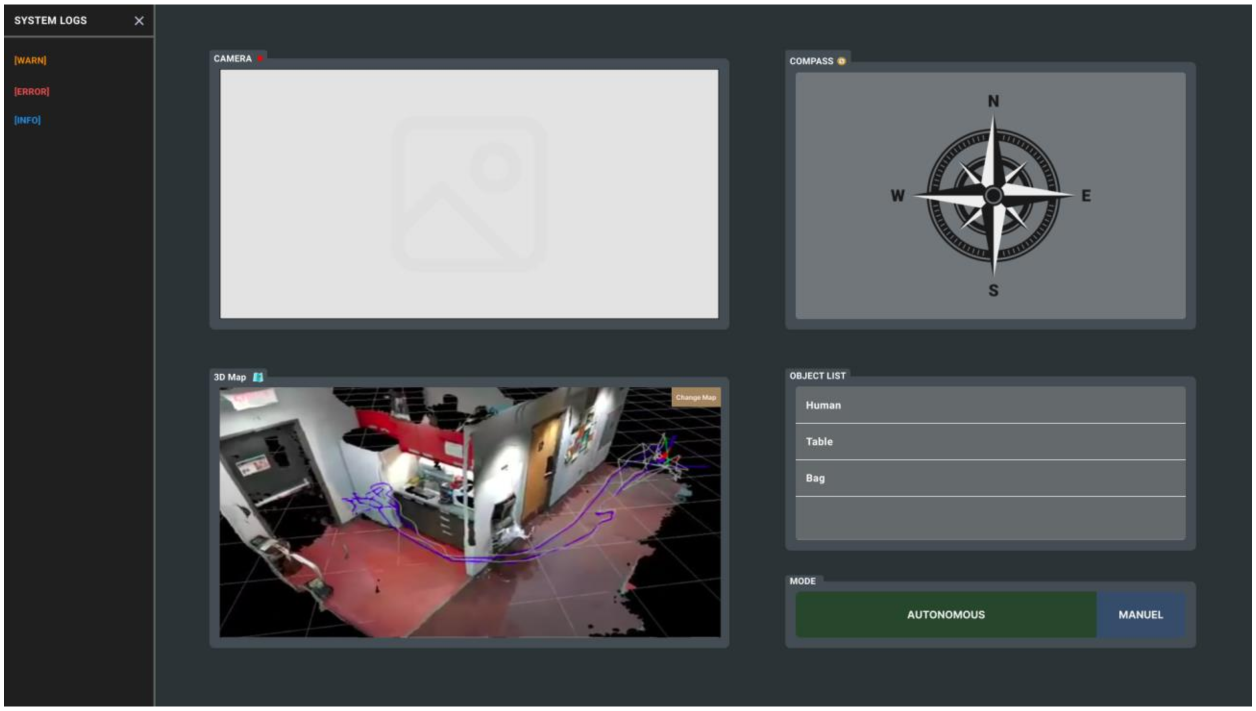
4.4.1. Home Page



4.4.2. Admin Dashboard with 2D Map



4.4.3. Admin Dashboard with 3D Map



4.4.4. Interactive UI Link

[Interactieve User Interface on Figma]

5. Project Work Plan

		Weeks															
Works	**Status**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Project Proposal Form	🟩	■	■														
Project Work Plan	🟩			■	■												
Literature Review	🟩				■	■	■										
SRS Document	🟩						■	■	■	■	■						
Project Website	🟩									■	■						
SDD Document	🟩											■	■	■			
Project Report	🟧														■		
Project Presentation	🟧															■	■

6. Conclusion

The Sentinel came up with this idea as a result of the use of unmanned aerial land vehicles, which are frequently encountered in convection wars in today's world. When unmanned vehicles are not used, nations lose many people. For this reason, unmanned vehicles are greatly needed in war zones. As a result, we decided to make an autonomous environmental mapping tool. However, our tool is a proof of concept and is not suitable for war zones. In line with this report, the necessary Literature Review preliminary studies were carried out for this idea to come to life, an SRS document was prepared to reveal these ideas more concretely for users and stakeholders and to determine various requirements, and an SDD document was prepared so that users and stakeholders can better perceive our design concepts and test our UI designs interactively. With our Project Report, which was formed by the combination of these documents, the groundwork for The Sentinel's intellectual approach was prepared and made ready to move on to the implementation phase.

References

[1] J. I.-G. You Li, "Lidar for Autonomous Driving: The principles, challenges, and trends for automotive lidar and perception systems," 17 April 2020.

[2] S. U. H. Syed, "ResearchGate," ResearchGate, 14 March 2022. [Online]. Available: https://www.researchgate.net/publication/359263639_Lidar_Sensor_in_Autonomous_Vehicles. [Accessed 6 November 2024].

[3] D. S. M. A. P. D. o. T. G. P. ,. D. o. E. G. M. Basavanna M, " An Overview of Path Planning and Obstacle Avoidance Algorithms in Mobile Robots," *International Journal of Engineering Research & Technology (IJERT),* 18 December 2019.

[4] N. S. C. D. J. E. S. Karur Karthik, "mdpi," 27 May 2021. [Online]. Available: https://www.mdpi.com/2624-8921/3/3/27. [Accessed 5 November 2024].

[5] "Robotic Path Planning," MIT, [Online]. Available: https://fab.cba.mit.edu/classes/865.21/topics/path_planning/robotic.html. [Accessed 6 November 2024].

[6] "A* search algorithm," Ada Computer Science, [Online]. Available: https://adacomputerscience.org/concepts/path_a_star. [Accessed 6 November 2024].

[7] "Path Planning Algorithms for robotic systems," [Online]. Available: https://roboticsbiz.com/path-planning-algorithms-for-robotic-systems/. [Accessed 6 November 2024].

[8] "D*," Wikipedia, [Online]. Available: https://roboticsbiz.com/path-planning-algorithms-for-robotic-systems/. [Accessed 6 November 2024].

[9] D. Biswas, "D*, D* Lite & LPA*," Medium, [Online]. Available: https://dibyendu-biswas.medium.com/d-d-lite-lpa-e7483779a7ca. [Accessed 6 November 2024].

[10] T. Chinenov, "Robotic Path Planning: RRT and RRT*," Medium, 14 February 2019. [Online]. Available: https://theclassytim.medium.com/robotic-path-planning-rrt-and-rrt-212319121378. [Accessed 6 November 2024].

[11] " Genetic Algorithms," GeeksforGeeks, 8 March 2024. [Online]. Available: https://www.geeksforgeeks.org/genetic-algorithms/. [Accessed 6 November 2024].

[12] "Ant colony optimization algorithms," Wikipedia, 22 September 2024. [Online]. Available: https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms. [Accessed 6 November 2024].

[13] A. N. A. K. B. B. M. H. M. N. ,. M. F. A. A. S. A. M. Mohd Nadhir Ab Wahab, "Plos One," 12 August 2024. [Online]. Available: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0308264. [Accessed 6 November 2024].

[14] "Robot Operating System," Wikipedia, 31 October 2024. [Online]. Available: https://en.wikipedia.org/wiki/Robot_Operating_System. [Accessed 6 November 2024].

[15] "Introduction to Loop Closure Detection in SLAM," Think Autonomous, 18 April 2023. [Online]. Available: https://www.thinkautonomous.ai/blog/loop-closure/. [Accessed 6 November 2024].

[16] T. B. H. Durrant-Whyte, "IEEE Xplore," June 2006. [Online]. Available: https://ieeexplore.ieee.org/document/1638022. [Accessed 6 November 2024].

[17] T. S. H. D. P. P. S. P. P Sankalprajan, "IEEE Xplore," 5 June 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9154101. [Accessed 5 November 2024].

[18] K. Libby, "What is Object Classification and How Can We Use It?," Shutterstock, 14 April 2023. [Online]. Available: https://www.shutterstock.com/blog/object-classification-in-computer-vision. [Accessed 6 November 2024].

[19] H. S. Yoshikatsu Nakajima, "IEEE Xplore," 16 December 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8579143. [Accessed 6 November 2024].

[20] B. K, "Object Detection Algorithms and Libraries," Neptune Ai, 15 April 2024. [Online]. Available: https://neptune.ai/blog/object-detection-algorithms-and-libraries. [Accessed 6 November 2024].

[21] "Yeong, D. J., Velasco-Hernandez, G., Barry, J., & Walsh, J. (2021). Sensor and sensor fusion technology in autonomous vehicles: A review. Sensors, 21(6), 2140.".

[22] "rtabmap," introlab, [Online]. Available: https://github.com/introlab/rtabmap. [Accessed 6 11 2024].

[23] "ROS," [Online]. Available: https://www.ros.org/. [Accessed 6 11 2024].

[24] "Project-2-Mapping," [Online]. Available: https://github.com/hungarianrobot/Project-2-Mapping. [Accessed 6 11 2024].

[25] "3D-Printed-ROS-SLAM-Robot," [Online]. Available: https://github.com/pliam1105/3D-Printed-ROS-SLAM-Robot. [Accessed 06 11 2024].

[26] "Medium," [Online]. Available: https://medium.com/@pliam1105/building-a-3d-printed-robot-that-uses-slam-for-autonomous-navigation-cd83473dac7c. [Accessed 06 11 2024].

[27] M. Labbe, “rtabmap_ros,” 10 December 2023. [Online]. Available: http://wiki.ros.org/rtabmap_ros. [Accessed 1 December 2024].

[28] "YOLO: Real-Time Object Detection," [Online]. Available: https://pjreddie.com/darknet/yolo/. [Accessed 1 December 2024].

[29] A. Anwar, "Create Your First ROS Publisher and Subscriber Nodes," Medium, 11 February 2021. [Online]. Available: https://medium.com/swlh/part-3-create-your-first-ros-publisher-and-subscriber-nodes-2e833dea7598. [Accessed 3 December 2024].