amblem, simge, sembol, logo, ticari marka içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

**ÇANKAYA UNIVERSITY**

**FACULTY OF ENGINEERING**

**COMPUTER ENGINEERING DEPARTMENT**

**Project Report**

**CENG 408**

Innovative System Design and Development II

**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](#_Toc198045786)

[2. Software Requirement Specification 2](#_Toc198045787)

[2.1. Introduction 2](#_Toc198045788)

[2.1.1. Purpose of This Document 2](#_Toc198045789)

[2.1.2. Scope of The Project 2](#_Toc198045790)

[2.1.3. Glossary 3](#_Toc198045791)

[2.2. Overall Description 4](#_Toc198045792)

[2.2.1. User Characteristics 4](#_Toc198045793)

[2.2.2. Product Perspective 5](#_Toc198045794)

[2.2.3. Product Functions 6](#_Toc198045795)

[2.3. Requirement Specification 7](#_Toc198045796)

[2.3.1. External Interface Requirements 7](#_Toc198045797)

[2.3.2. Functional Requirements 13](#_Toc198045798)

[2.3.3. Non-Functional Requirements 22](#_Toc198045799)

[2.4. Use Cases 25](#_Toc198045800)

[2.4.1. Movement 25](#_Toc198045801)

[2.4.2. Data Stream 25](#_Toc198045802)

[2.4.3. Manual Mapping 26](#_Toc198045803)

[2.4.4. Autonomous Mapping 27](#_Toc198045804)

[2.4.5. Object Detection 27](#_Toc198045805)

[3. Software Design Document 28](#_Toc198045806)

[3.1. Introduction 28](#_Toc198045807)

[3.1.1. Purpose of this Document 28](#_Toc198045808)

[3.1.2. Scope of this Document 28](#_Toc198045809)

[3.1.3. Glossary 29](#_Toc198045810)

[3.2. Overview of Document 29](#_Toc198045811)

[3.3. System Design 30](#_Toc198045812)

[3.3.1. Architectural Design 30](#_Toc198045813)

[3.3.2. Data Flow Diagrams 30](#_Toc198045814)

[3.3.3. Class Diagrams 33](#_Toc198045815)

[3.3.4. Activity Diagrams 35](#_Toc198045816)

[3.3.5. Sequence Diagrams 40](#_Toc198045817)

[3.4. User Interfaces 44](#_Toc198045818)

[3.4.1. Home Page 44](#_Toc198045819)

[3.4.2. Admin Dashboard with 2D Map 44](#_Toc198045820)

[3.4.3. Admin Dashboard with 3D Map 45](#_Toc198045821)

[3.4.4. Interactieve UI Link 45](#_Toc198045822)

[4. Test Plan & Result Documents 46](#_Toc198045823)

[4.1 Introduction 46](#_Toc198045824)

[4.1.1. Version Control 46](#_Toc198045825)

[4.1.2. Overview 46](#_Toc198045826)

[4.1.3. Scope of The Document 46](#_Toc198045827)

[4.1.4. Terminology 46](#_Toc198045828)

[4.2 Features to Be Tested: 47](#_Toc198045829)

[4.2.1. Web UI 47](#_Toc198045830)

[4.2.2. Hardware 48](#_Toc198045831)

[4.2.3. Manual Movement 48](#_Toc198045832)

[4.2.4. Autonomous Movement 49](#_Toc198045833)

[4.2.5. Manual Mapping 49](#_Toc198045834)

[4.2.6. Autonomous Mapping 50](#_Toc198045835)

[4.2.7. Object Detection 50](#_Toc198045836)

[4.2.8. Simulation 51](#_Toc198045837)

[4.3. Detailed Test Cases 52](#_Toc198045838)

[4.3.1 Web UI 52](#_Toc198045839)

[4.3.2. Hardware 57](#_Toc198045840)

[4.3.3. Manual Movement 59](#_Toc198045841)

[4.3.4. Autonomous Movement 61](#_Toc198045842)

[4.3.5. Manual Mapping 61](#_Toc198045843)

[4.3.6. Autonomous Mapping 65](#_Toc198045844)

[4.3.7. Object Detection 67](#_Toc198045845)

[4.3.8 Simulation 70](#_Toc198045846)

[4.4. Features To Be Not Tested 72](#_Toc198045847)

[4.4.1. Windows Operating System Compatibility 72](#_Toc198045848)

[4.4.2. Driving on Flat Surfaces Only 72](#_Toc198045849)

[4.4.3. Well-Lit Environment 72](#_Toc198045850)

[4.4.4. Limited to the ROS-Jazzy and Gazebo Harmonic Versions 72](#_Toc198045851)

[4.4.5. Internet Connectivity and Real Time Data 72](#_Toc198045852)

[4.5. Pass/Fail Criteria 73](#_Toc198045853)

[4.5.1. Manual Movement 73](#_Toc198045854)

[4.5.2. Autonomous Movement 73](#_Toc198045855)

[4.5.3. Manual Mapping 73](#_Toc198045856)

[4.5.4. Autonomous Mapping 73](#_Toc198045857)

[4.5.5. Object Detection 73](#_Toc198045858)

[4.5.6. Simulation 73](#_Toc198045859)

[4.6. Exit Criteria 73](#_Toc198045860)

[4.7. Test Results 74](#_Toc198045861)

[4.8. Summary of the Test Results 77](#_Toc198045862)

[References 78](#_Toc198045863)

# Introduction

This project report prepared for The Sentinel: Discovery Autonomous Vehicle includes materials, contents, and working reports related to the Software Requirement Specification, and Software Design Document. 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. Under the Test Plan & Results section provides us with a plan about the Test Cases related to the main feature of the project. We will outline the features that are not to be tested, as well as our success, failure, and exit criteria. Also give the Summary of the Test Results to show the providing the Exit Criteria. This project report, created by bringing together these three separate documents, provides information about the purpose and development stages of The Sentinel project.

# Software Requirement Specification

## 2.1. Introduction

### 2.1.1. Purpose of This Document

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.

### 2.1.2. 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 [1], 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 also plan to detect the objects using YOLO, [2] a library for object detection, and then store these informations using rosbag [3].

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.

### 2.1.3. 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 that helps you control the motors. |
| 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 |
| Power Bank | 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 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. |
| Slam\_toolbox | Slam Toolbox is a set of tools and capabilities for 2D SLAM. |
| 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. |

## 

## 2.2. Overall Description

### 2.2.1. User Characteristics

#### 2.2.1.1. Operators

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.

#### 2.2.1.2. 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.

### 2.2.2. 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 Li-On 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 2.3.1.2. Hardware Interfaces section and software interface will be given at 2.3.1.3.

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** |
| Manuel 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 |

### 2.2.3. Product Functions

**Manuel Movement:** The system is designed to performs directional movements based on user i̇nputs (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 is generated with the slam\_toolbox package which is integrated within ROS2 and RViz, to generate the 2D grid. slam\_toolbox incorporates information from laser scanners in the form of a LaserScan message and TF transforms from odom->base link, and creates a map 2D map of a space.

**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.

## 2.3. Requirement Specification

### 2.3.1. External Interface Requirements

#### 2.3.1.1. 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 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 visualize the vehicle model on the Admin Dashboard panel, and according to the moving the car the model also will move that vehicle direction. The interface section is constructed 4 separated boxes which is included Live Camera, Laser Scan, 2D and 3D Map, 3D Vehicle Model, and Joystick or Keyboard options to move manually and their animations. Users 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.

#### 2.3.1.2. 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 Use** |
|  | 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. |

#### 2.3.1.3. 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.

##### 2.3.1.3.1. 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.

##### 2.3.1.3.2. Movement 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.

##### 2.3.1.3.3. 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.

#### 2.3.1.4. 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. [4]

### 2.3.2. 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** | Scenario 1:   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   Scenario 2:   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. In this situation hardware problem should be fixed. |

|  |  |
| --- | --- |
| **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. In this situation the connection should be re-established |

|  |  |
| --- | --- |
| **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** | The connection between the Remote Computer System and Sentinel might be lost. The data transfer stops until the connection is re-established between the Remote Computer System and Sentinel. In this situation the connection should be re-established |

|  |  |
| --- | --- |
| **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 Web Application might be lost. The data transfer stops until the connection is re-established between Remote Computer System and Web. In this situation the connection should be re-established. |

|  |  |
| --- | --- |
| **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. In this situation web server should be restarted. |

|  |  |
| --- | --- |
| **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 detection objects. |
| **Scenario** | 1. The user starts the using system at the remote computer 2. Users choose the object detection mode. 3. Users can see the detection objects |
| **Exceptional Situations & Alternative Flow** | The object detection algorithms cannot understand the objects from images properly. Therefore, user cannot see the results. |

### 2.3.3. Non-Functional Requirements

#### 

#### 2.3.3.1 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. |

#### 

#### 2.3.3.2 Safety Requirements

|  |  |
| --- | --- |
| **Safety Requirements** | **Description** |
| Damage Preventing | The vehicle must make quick and effective manoeuvres 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. |

#### 2.3.3.3 Security Requirements

|  |  |
| --- | --- |
| **Security Requirements** | **Description** |
| Wi-Fi Access | The connected Wi-Fi 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. |

#### 

#### 2.3.3.4 Software Quality Attributes

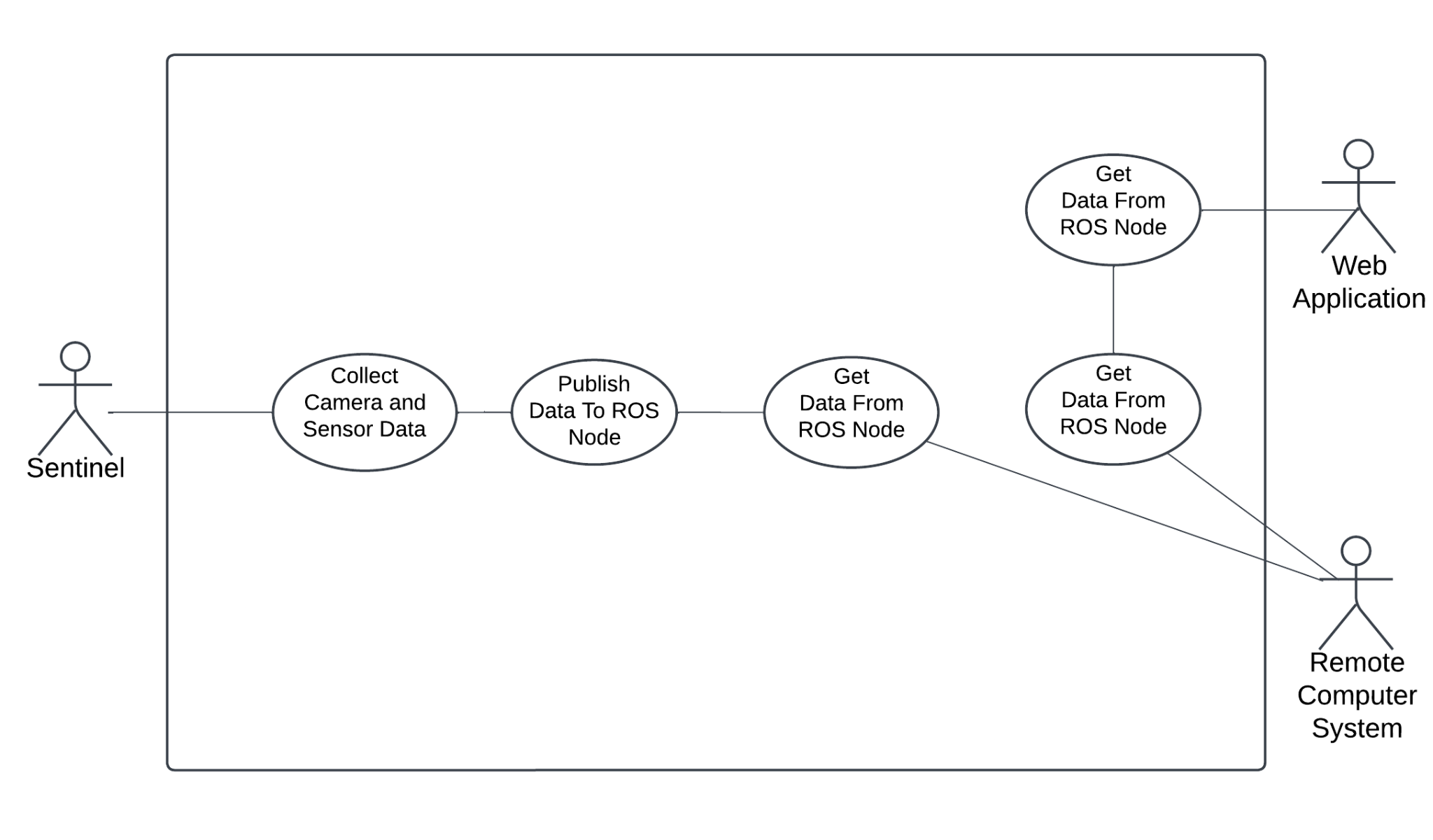
|  |  |
| --- | --- |
| **Software Quality Attributes** | **Description** |
| Reliability | The system shouldn’t produce errors in normal conditions and complete the task successfully. |
| Robustness | Under heavy environmental conditions the vehicle must be able to complete its tasks. |
| Portability | 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. |

## 2.4. Use Cases

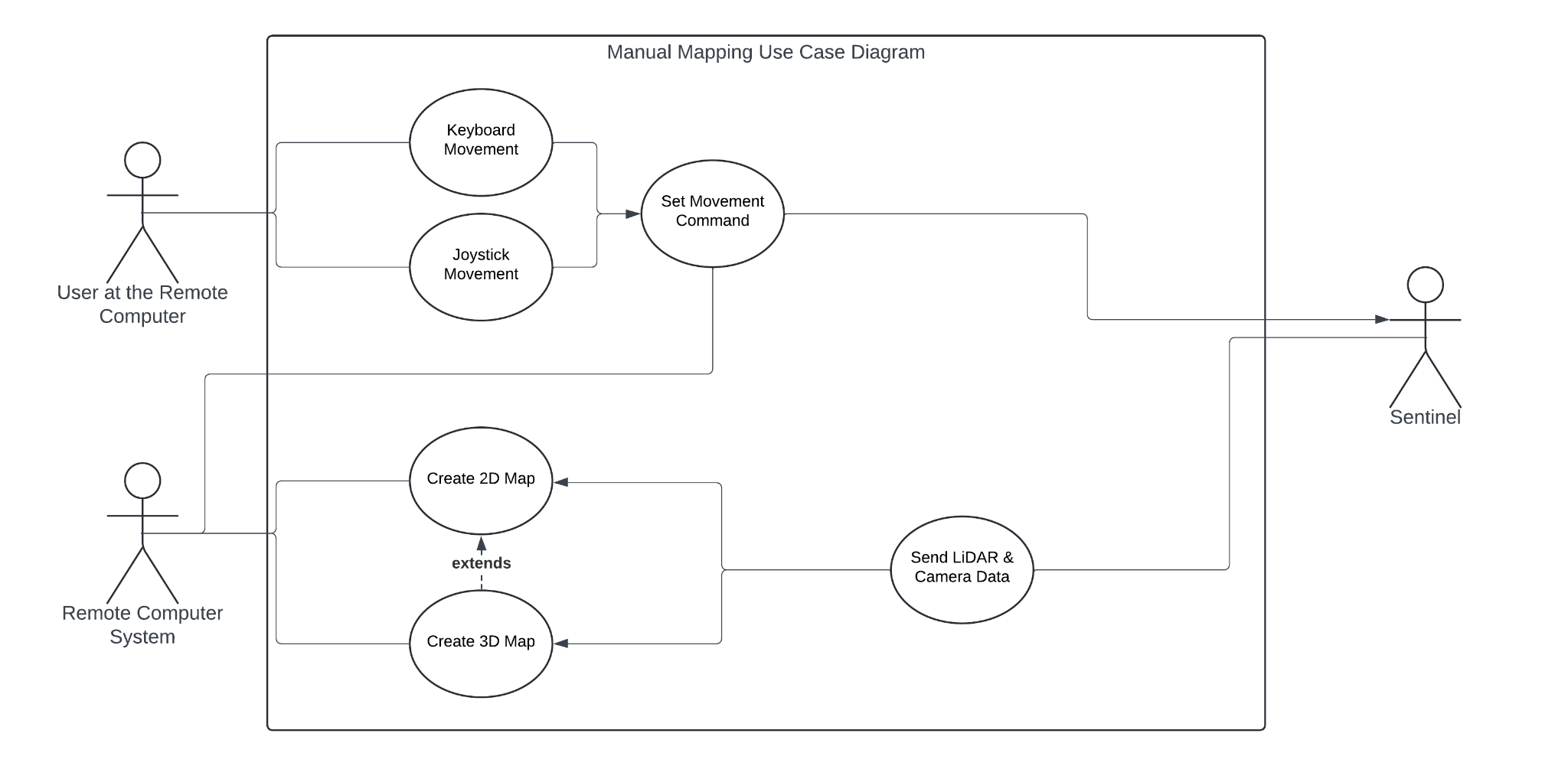
### 2.4.1. Movement

### 

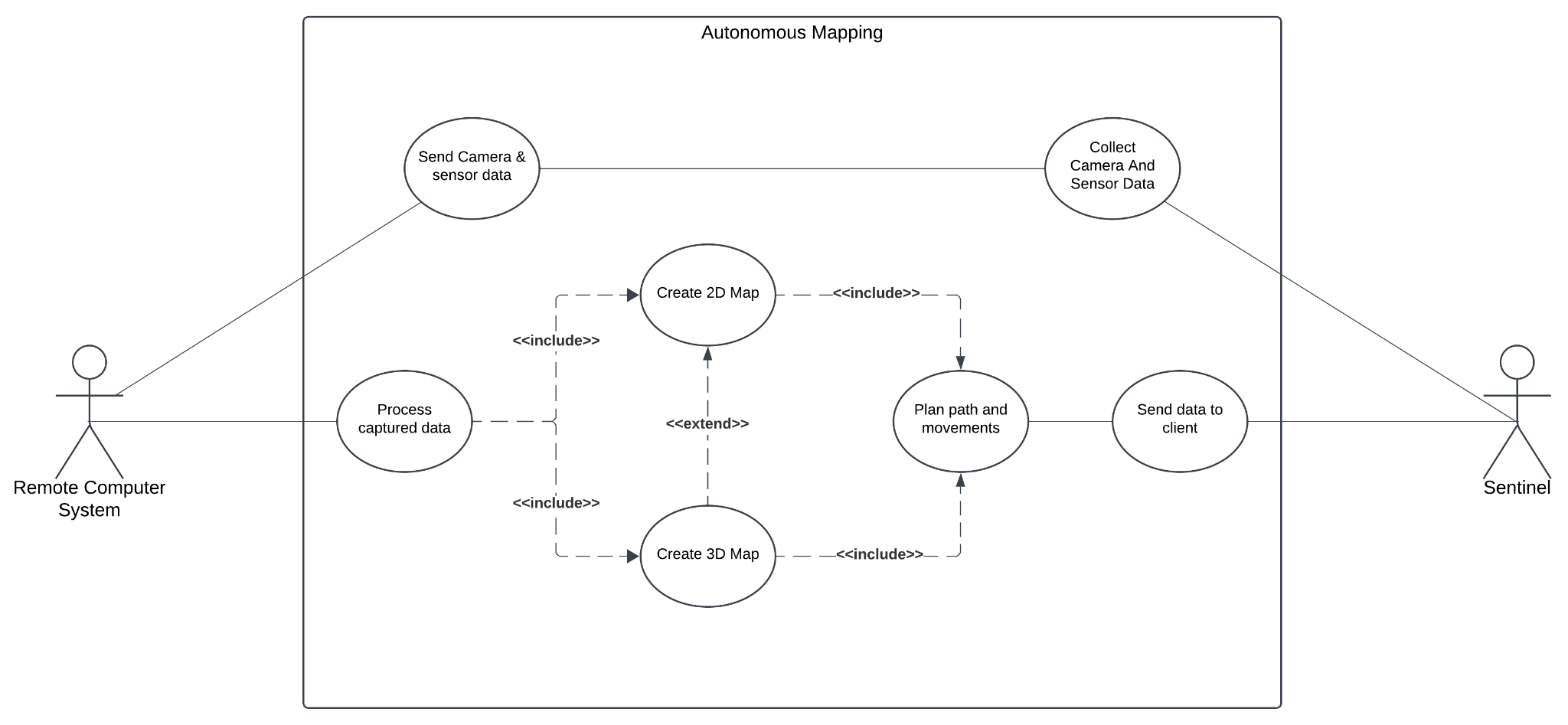
### 2.4.2. Data Stream



### 2.4.3. Manual Mapping



### 2.4.4. Autonomous Mapping



### 2.4.5. Object Detection

metin, diyagram, taslak, çizim içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

# 3. Software Design Document

## 3.1. Introduction

### 3.1.1. 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 the 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.

### 3.1.2. 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.

### 3.1.3. 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 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. |

## 3.2. Overview of 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.

## 3.3. System Design

### 3.3.1. Architectural Design

metin, diyagram, paralel, çizgi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

### 3.3.2. Data Flow Diagrams

#### 3.3.2.1. Context Diagram

metin, yazı tipi, çizgi, diyagram içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

#### 3.3.2.2. Level 1 DFDs

##### 3.3.2.2.1. Movement

##### metin, diyagram, daire, paralel içeren bir resim Yapay zeka tarafından oluşturulan içerik yanlış olabilir.3.3.2.2.2. Data Stream

#### metin, diyagram, yazı tipi, çizgi içeren bir resim Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

##### 

##### 3.3.2.2.3. Manual Mapping

diyagram, taslak, daire, çizgi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

##### 3.3.2.3.4. Autonomous Mapping

metin, diyagram, daire, çizgi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

##### 3.3.2.3.5. Object Detection

metin, diyagram, daire, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

### 3.3.3. Class Diagrams

#### 3.3.3.1 Class Diagram for Remote Computer

metin, diyagram, ekran görüntüsü, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

#### 3.3.3.2 Class Diagram for Vehicle (Raspberry Pi)

metin, diyagram, yazı tipi, doküman, belge içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

### 3.3.4. Activity Diagrams

#### 3.3.4.1. Movementmetin, ekran görüntüsü, diyagram, yazı tipi içeren bir resim Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

#### 3.3.4.2 Data Stream

metin, diyagram, taslak, çizim içeren bir resim

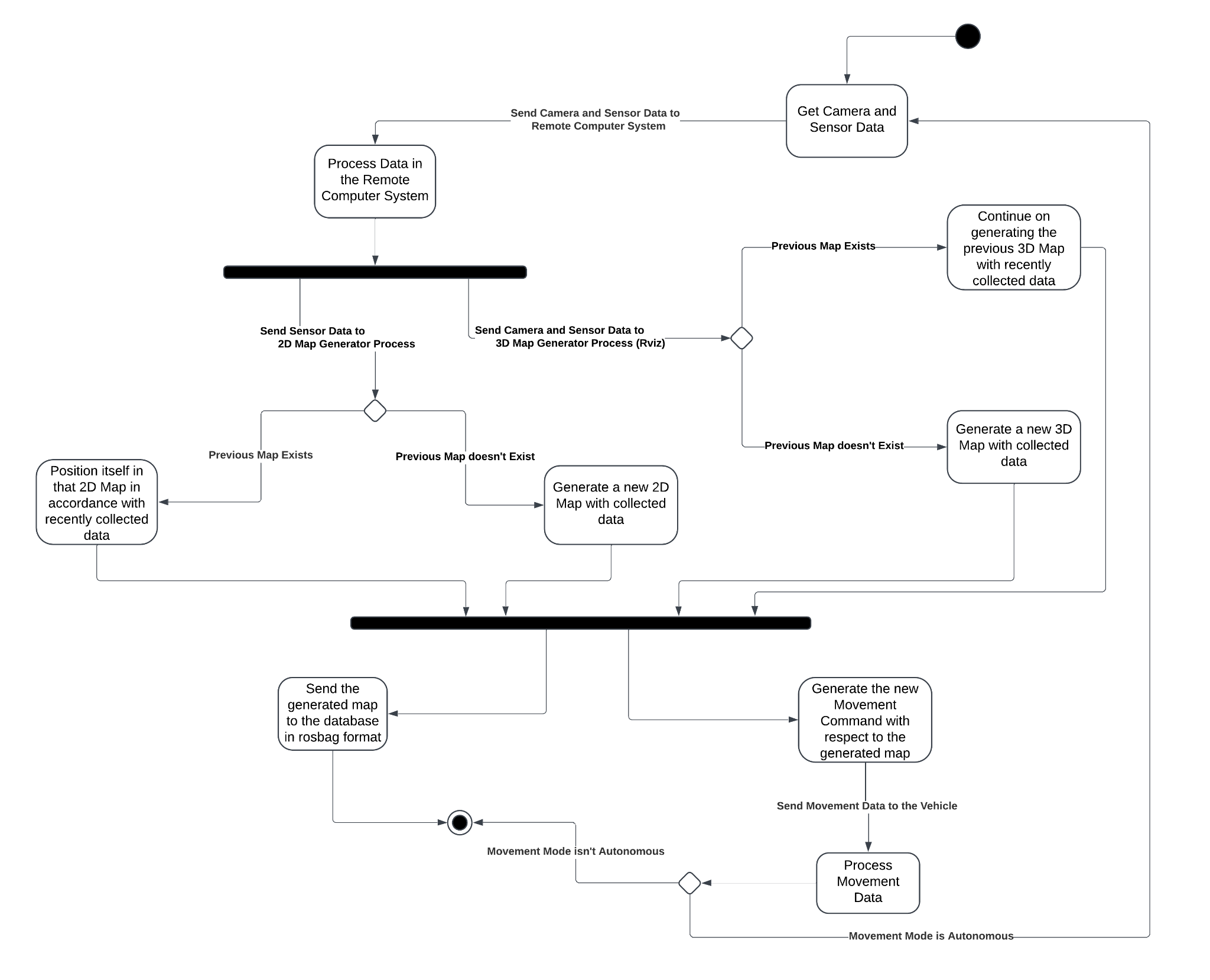
Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

#### 3.3.4.3. Manual Mapping

metin, diyagram, ekran görüntüsü, paralel içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

#### 3.3.4.4. Autonomous Mapping



#### 3.3.4.5. Object Detection

metin, diyagram, ekran görüntüsü, taslak içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

### 3.3.5. Sequence Diagrams

#### 3.3.5.1. Movementmetin, diyagram, plan, teknik çizim içeren bir resim Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

### 

### 

### 

### 

### 

### 

### 

#### 3.3.5.2. Data Stream

metin, diyagram, plan, teknik çizim içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

#### 3.3.5.3. Manual Mapping

metin, diyagram, plan, teknik çizim içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

#### 

#### 3.3.5.4. Autonomous Mapping

metin, diyagram, plan, teknik çizim içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

#### 

#### 3.3.5.5. Object Detection

metin, diyagram, plan, teknik çizim içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

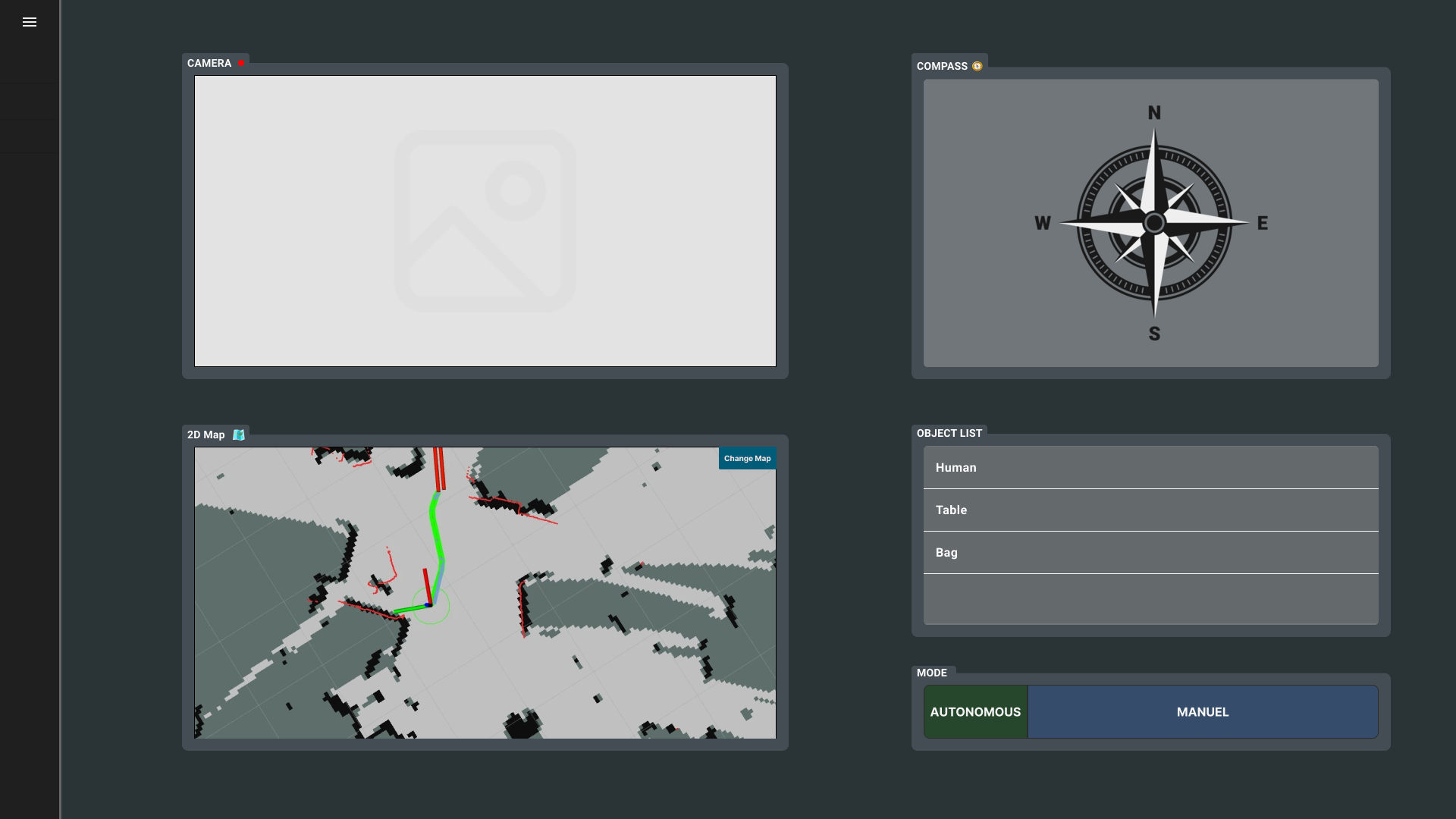
## 3.4. User Interfaces

### 3.4.1. Home Page

tekerlek, araba lastiği, taşımak, nakletmek, taşıt, araç içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

### 3.4.2. Admin Dashboard with 2D Map



### 3.4.3. Admin Dashboard with 3D Map

ekran görüntüsü, metin, multimedya yazılımı, grafik yazılımı içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

### 3.4.4. Interactieve UI Link

[Interactieve User Interface on Figma](https://www.figma.com/proto/XlJogvLIsypZYikcVeXhkN/Sentinel%3A-Autonomous-Discovery-Vehicle?node-id=678-1957&p=f&t=aVYLa2LtVc4KJ8Kf-1&scaling=scale-down&content-scaling=fixed&starting-point-node-id=678%3A1957)

# Test Plan & Result Documents

## 4.1 Introduction

### 4.1.1. Version Control

|  |  |  |
| --- | --- | --- |
| **Version Number** | **Description of Changes** | **Date** |
| sentinel@1.0.0 | First version including the autonomous movement and mapping in the simulation | 27 March 2025 |

### 4.1.2. Overview

Sentinel: Autonomous Discovery Vehicle is an autonomous discovery vehicle that maps its environment in 2D and 3D using SLAM algorithms. Use cases and software design are detailed in the Software Requirements Specification (SRS) and Software Design Description (SDD) documents. This test plan outlines procedures for evaluating the vehicle's performance against these specified specifications in an unknown environment. It defines the test methodology, success and exit criteria, and expected results to ensure the system meets intended functionality and reliability standards.

### 4.1.3. Scope of The Document

This document will provide a detailed explanation of the test cases. Additionally, we will outline the features that are not to be tested, as well as our success, failure, and exit criteria. Finally, we will present the test results to assess the overall performance of the system.

### 4.1.4. Terminology

|  |  |
| --- | --- |
| **Acronym** | **Definition** |
| SRS | Software Requirements Specification |
| SDD | Software Design Document |
| UI | User Interface |
| 2D | Two-dimensional |
| 3D | Three-dimensional |
| ROS | Robot Operating System |
| YOLO | You Look Only Once (Object Detection Framework) |
| LIDAR (Light detection and Ranging) Sensor | A sensor for determining ranges by targeting an object or a surface with a laser and measuring the time for the reflected to return to the receiver. |
| MUI | Material UI |

## 4.2 Features to Be Tested:

### 4.2.1. Web UI

Web UI processes while The Sentinel activities showing on the web application as an Admin dashboard format.

|  |  |  |  |
| --- | --- | --- | --- |
| **TC\_ID** | **Requirements** | **Priority** | **Scenario** |
| WUI.01 | Node.js, NPM, Wi-Fi connection, Joystick | Low | The user can control the Sentinel with a joystick and see its movements simultaneously in the web browser. |
| WUI.02 | Node.js, Wi-Fi connection, Keyboard | Low | The user can control the Sentinel with a keyboard and see the pressed keys simultaneously in the web browser. |
| WUI.03 | Node.js, NPM, Wi-Fi connection | Low | The user can view the real-time video captured by the camera on the web UI, pause it, and switch to full-screen mode. |
| WUI.04 | Node.js, NPM, Wi-Fi connection | Low | The user can see the real-time generation of the 2D map on the web UI, switch to full-screen mode, and download the generated map. |
| WUI.05 | Node.js, NPM, Wi-Fi connection | Low | The user can view the vehicle's movement direction through the 3D vehicle model in the web browser. |
| WUI.06 | Node.js, NPM, Wi-Fi connection | Low | The user can see the 3D Map after it is completed. |
| WUI.07 | Node.js, NPM, Wi-Fi connection | Low | The user can display the system logs by clicking the hamburger menu button located at the top left of the screen. |

### 4.2.2. Hardware

Hardware processes while The Sentinel is prepared for real life applications.

|  |  |  |  |
| --- | --- | --- | --- |
| **TC\_ID** | **Requirements** | **Priority** | **Scenario** |
| H.01 | Wires, Tires, Motors, Car Chassis, Motor Driver, Battery, Powerbank, Raspberry Pi | Low | The user can control the Sentinel without any loose contact in cables. The Sentinel must move to the desired direction instantly. |
| H.02 | Wires, Tires, Motors, Car Chassis, Motor Driver, Battery, Powerbank, Raspberry Pi | Low | The motor driver can deliver the required voltage to motors for the movement with respect to given direction. |
| H.03 | Wires, Tires, Motors, Car Chassis, Motor Driver, Battery, Powerbank, Raspberry Pi | Low | All of the motors can be turned to the desired way while Sentinel is controlling. (forward or backward) |
| H.04 | Wires, Tires, Motors, Car Chassis, Motor Driver, Battery, Powerbank, Raspberry Pi | Low | The Sentinel’s Lithium-Ion battery can deliver 1.5 hours of continuous operation at maximum power. |
| H.05 | Wires, Tires, Motors, Car Chassis, Motor Driver, Battery, Powerbank, Raspberry Pi | Low | The Sentinel’s powerbank can deliver 8 hours of continuous operation at maximum power |

### 4.2.3. Manual Movement

Manual Movement processes while The Sentinel is controlled by a user from a joystick or keyboard in a simulation or real life.

|  |  |  |  |
| --- | --- | --- | --- |
| **TC\_ID** | **Requirements** | **Priority** | **Scenario** |
| MMV.01 | Joystick (or Keyboard), Hardware Components | High | The Manual Movement will be processed while Sentinel is moving with controlled by the user, using the joystick (or keyboard) in real life. |
| MMV.02 | Joystick (or Keyboard), Car Model, ROS Visualization Apps | Medium | The Manual Movement will be processed while Sentinel is moving with controlled by the user, using the joystick (or keyboard) in the simulation. |

### 4.2.4. Autonomous Movement

Autonomous Movement processes while The Sentinel is controlled by exploration and decision-making algorithms, in a simulation or in real life.

|  |  |  |  |
| --- | --- | --- | --- |
| **TC\_ID** | **Requirements** | **Priority** | **Scenario** |
| AMV.01 | Lidar, Hardware Components, Exploring Algorithms | Medium | The Autonomous Movement will be processed while Sentinel is moving without being controlled by the user, using the exploration of the objects, and deciding the path around the real-life environment. |
| AMV.02 | Lidar, Exploring Algorithms, Car Model, ROS Visualization Apps | Medium | The Autonomous Movement will be processed while Sentinel is moving without being controlled by the user, using the exploration of the objects, and deciding the path around the simulation environment. |

### 4.2.5. Manual Mapping

2D and 3D mapping process while the user drives the Sentinel manually from the joystick.

|  |  |  |  |
| --- | --- | --- | --- |
| **TC\_ID** | **Requirements** | **Priority** | **Scenario** |
| MM.2D.01 | Joystick, Lidar | Low | The 2D map will be created while Sentinel is moving with user control in a straight line without objects in simulation. |
| MM.2D.02 | Joystick, Lidar | Medium | The 2D map will be created while Sentinel is moving with user control in different directions without objects in simulation. |
| MM.2D.03 | Joystick, Lidar | Medium | The 2D map will be created while Sentinel is moving with user control in a straight line with an object in simulation. |
| MM.2D.04 | Joystick, Lidar | High | The 2D map will be created while Sentinel is moving around with user control with different objects in simulation. |
| MM.2D.05 | Joystick, Lidar | High | The 2D map will be created while Sentinel is moving around with user control with different objects in the real room. |

|  |  |  |  |
| --- | --- | --- | --- |
| **TC\_ID** | **Requirements** | **Priority** | **Scenario** |
| MM.3D.01 | Joystick, Lidar, Camera | Medium | The 3D map will be created while Sentinel is moving around with user control with different objects in simulation. |
| MM.3D.02 | Joystick, Lidar, Camera | Low | The 3D map will be created while Sentinel is moving around with user control with different objects in the real room. |

### 4.2.6. Autonomous Mapping

2D and 3D mapping process while the Sentinel is controlled by the autonomous algorithms.

|  |  |  |  |
| --- | --- | --- | --- |
| **TC\_ID** | **Requirements** | **Priority** | **Scenario** |
| AM.2D.01 | Joystick, Lidar, Autonomous Movement | High | The 2D map will be created while Sentinel is moving, using an autonomous movement algorithm in a simulation. |
| AM.2D.02 | Joystick, Lidar, Autonomous Movement | High | The 2D map will be created while Sentinel is moving, using an autonomous movement algorithm in a real room. |

|  |  |  |  |
| --- | --- | --- | --- |
| **TC\_ID** | **Requirements** | **Priority** | **Scenario** |
| AM.3D.01 | Joystick, Lidar, Camera, Autonomous Movement | High | The 3D map will be created while Sentinel is moving, using an autonomous movement algorithm in a simulation. |
| AM.3D.02 | Joystick, Lidar, Camera, Autonomous Movement | Low | The 3D map will be created while Sentinel is moving, using an autonomous movement algorithm in a real room. |

### 4.2.7. Object Detection

Object Detection processes with using real-time camera data.

|  |  |  |  |
| --- | --- | --- | --- |
| **TC\_ID** | **Requirements** | **Priority** | **Scenario** |
| OD.01 | Camera, Movement, YOLO model | Medium | Object detection algorithms will be actively performed to identify and detect objects in the simulation world. |
| OD.02 | Camera, Movement, YOLO model. | Medium | Object detection algorithms will be actively performed to identify and detect objects in the real world. |
| OD.03 | Camera, Movement, YOLO model. | Medium | The system will identify and distinguish multiple objects in an environment, each with unique characteristics like shape, size, or color, and track them as they interact. |
| OD.04 | Camera, Movement, YOLO model. | Medium | The system will detect objects at varying distances, adjusting its focus to accurately identify them. |
| OD.05 | Camera, Movement, YOLO model. | Medium | Evaluate the accuracy of detected objects. |
| OD.06 | Camera, Movement, YOLO model. | Medium | Object detection algorithm meets real-time performance requirements during movement |

### 4.2.8. Simulation

|  |  |  |  |
| --- | --- | --- | --- |
| **TC\_ID** | **Requirements** | **Priority** | **Scenario** |
| S.01 | Remote Computer, ROS2 Jazzy Harmonic | High | The movement of the Sentinel must be the same on RViz and Gazebo. |
| S.02 | Remote Computer, ROS2 Jazzy Harmonic | High | The Sentinel can create a 2D map of the Gazebo simulation world. |
| S.03 | Remote Computer, ROS2 Jazzy, Gazebo Harmonic | High | The Sentinel can create a 3D map of the Gazebo simulation world. |
| S.04 | Remote Computer, ROS2 Jazzy, Gazebo Harmonic | High | The Sentinel can move by avoiding objects on the Gazebo simulation world |
| S.05 | Remote Computer, ROS2 Jazzy, Gazebo Harmonic | High | The Sentinel can move autonomously to a published point on the Gazebo simulation world. |

## 

## 4.3. Detailed Test Cases

### 4.3.1 Web UI

|  |  |
| --- | --- |
| **TC\_ID** | WUI.01 |
| Purpose | Viewing real-time joystick controls on the web dashboard. |
| Requirements | Node.js, NPM, Wi-Fi connection, Joystick |
| Priority | Low |
| Estimated Time Needed | 1 hour |
| Dependency | Movement data from the ROS topic |
| Setup | Run the React Application, have Wi-Fi Connection, Connect to Rosbridge server |
| Procedure | 1. Open the web dashboard  2. Use the Joystick to publish data on movement topic  3. Observe whether the joystick on the web moves exactly the same as the physical one. |
| Cleanup | Close the web dashboard |

## 

|  |  |
| --- | --- |
| TC\_ID | WUI.02 |
| Purpose | Viewing real-time keyboard controls on the web dashboard. |
| Requirements | Node.js, NPM, Wi-Fi connection, Keyboard |
| Priority | Low |
| Estimated Time Needed | 1 hour |
| Dependency | Movement data from the ROS topic |
| Setup | Run the React Application, have Wi-Fi Connection, Connect to Rosbridge server |
| Procedure | 1. Open the web dashboard  2. Use the keyboard to publish data on movement topic  3. Observe whether the keyboard on the web moves exactly the same as the physical one. |
| Cleanup | Close the web dashboard |

|  |  |
| --- | --- |
| **TC\_ID** | WUI.03 |
| Purpose | Viewing the real-time video footage on the web dashboard |
| Requirements | Node.js, NPM, Wi-Fi connection |
| Priority | Low |
| Estimated Time Needed | 1 hour |
| Dependency | Camera data from the ROS topic, a stable internet connection |
| Setup | Run the React Application, have Wi-Fi Connection, Connect to Rosbridge server |
| Procedure | 1. Open the web dashboard  2. Publish the camera data to camera topic from Sentinel  3. Click on the play button of the camera container on web dashboard  4. Observe whether the camera data arrives in real-time.  5. Click on the expand button to see the camera in full-screen.  6. Pause the camera |
| Cleanup | Close the web dashboard |

|  |  |
| --- | --- |
| **TC\_ID** | WUI.04 |
| Purpose | Viewing the generated 2D Map on the web dashboard |
| Requirements | Node.js, NPM, Wi-Fi connection |
| Priority | Low |
| Estimated Time Needed | 2 hour |
| Dependency | Map data from the ROS topic |
| Setup | Run the React Application, have Wi-Fi Connection, Connect to Rosbridge server |
| Procedure | 1. Open the web dashboard  2. Publish the map data to map topic from remote computer system  3. Click on the play button of the 2D Map container on web dashboard  4. Observe whether the 2D Map data arrives in real-time.  5. Click on the expand button to see the 2D Map in full-screen.  6. Pause the 2D Map  7. Download the generated 2D Map |
| Cleanup | Close the web dashboard |

|  |  |
| --- | --- |
| **TC\_ID** | WUI.05 |
| Purpose | Viewing the Sentinel’s movement direction on the web dashboard in real-time |
| Requirements | Node.js, NPM, Wi-Fi connection |
| Priority | Low |
| Estimated Time Needed | 2 hours |
| Dependency | Movement data from the ROS topic, (Keyboard, Joystick, and autonomous Movement) |
| Setup | Run the React Application, have Wi-Fi Connection, Connect to Rosbridge server |
| Procedure | 1. Open the web dashboard  2. Publish the movement data to map topic from remote computer system  3. Observe whether the 3D Model moves at the same direction as the Joystick or Keyboard movement. |
| Cleanup | Close the web dashboard |

|  |  |
| --- | --- |
| **TC\_ID** | WUI.06 |
| Purpose | Viewing the 3D Map after the mapping is completed |
| Requirements | Node.js, NPM, Strong Wi-Fi connection |
| Priority | Low |
| Estimated Time Needed | 2 hours |
| Dependency | Generated 3D Map from Rtabmap. |
| Setup | Run the React Application, have Wi-Fi Connection, Connect to Rosbridge server |
| Procedure | 1. Open the web dashboard  2. Click on the generate 3D Map button in the 3D Map container section  3. Observe whether the 3D Map is generated |
| Cleanup | Close the web dashboard |

|  |  |
| --- | --- |
| **TC\_ID** | WUI.07 |
| Purpose | Viewing the active system logs |
| Requirements | Node.js, NPM, Wi-Fi connection |
| Priority | Low |
| Estimated Time Needed | 1 hour |
| Dependency | No dependencies |
| Setup | Run the React Application, have Wi-Fi Connection, Connect to Rosbridge server |
| Procedure | 1. Open the web dashboard  2. Click on the generate hamburger menu button located at the top left of the screen.  3. Do an operation (Start Camera, Move the vehicle, Download 2D Map) or see whether an object is detected and printed in the logs |
| Cleanup | Close the web dashboard |

### 4.3.2. Hardware

|  |  |
| --- | --- |
| **TC\_ID** | **H.01** |
| Purpose | Controlling the Sentinel |
| Requirements | Wires, Tires, Motors, Car Chassis, Motor Driver, Battery, Powerbank, Raspberry Pi |
| Priority | Low |
| Estimated Time Needed | 5 minutes |
| Dependency | Well connected wires. |
| Setup | Give power to motors and Raspberry Pi. Connect the joystick. |
| Procedure | 1. Run the manual movement package on both Raspberry Pi and Remote Computer. 2. Control the car via joystick or keyboard 3. Check the movement of the car |
| Cleanup | Interrupt both packages. Power off the battery and Raspberry Pi. |

|  |  |
| --- | --- |
| **TC\_ID** | **H.02** |
| Purpose | Controlling the Sentinel |
| Requirements | Wires, Tires, Motors, Car Chassis, Motor Driver, Battery, Powerbank, Raspberry Pi |
| Priority | Low |
| Estimated Time Needed | 5 minutes |
| Dependency | Well connected wires. |
| Setup | Give power to motors and Raspberry Pi. Connect the joystick. |
| Procedure | 1. Run the manual movement package on both Raspberry Pi and Remote Computer. 2. Drive the Sentinel via joystick or keyboard 3. Check the voltage values on motor driver via voltmeter |
| Cleanup | Interrupt both packages. Power off the battery and Raspberry Pi. |

|  |  |
| --- | --- |
| **TC\_ID** | **H.03** |
| Purpose | Controlling the Sentinel |
| Requirements | Wires, Tires, Motors, Car Chassis, Motor Driver, Battery, Powerbank, Raspberry Pi |
| Priority | Low |
| Estimated Time Needed | 5 minutes |
| Dependency | Well connected wires. |
| Setup | Give power to motors and Raspberry Pi. Connect the joystick. |
| Procedure | 1. Run the manual movement package on both Raspberry Pi and Remote Compuer. 2. Drive the car in backward and forward direction. 3. Check the movement of all motors |
| Cleanup | Interrupt both packages. Power off the battery and Raspberry Pi. |

|  |  |
| --- | --- |
| **TC\_ID** | **H.04** |
| Purpose | Powering the motors |
| Requirements | Wires, Tires, Motors, Car Chassis, Motor Driver, Battery, Powerbank, Raspberry Pi |
| Priority | Low |
| Estimated Time Needed | 1.5 Hours |
| Dependency | Fully charged battery. |
| Setup | Give power to motors and Raspberry Pi. |
| Procedure | 1. Run the desired package that uses motors on the system. 2. Check the time |
| Cleanup | Interrupt packages. Power off the battery and Raspberry Pi. |

|  |  |
| --- | --- |
| **TC\_ID** | **H.05** |
| Purpose | Powering the Raspberry Pi |
| Requirements | Wires, Tires, Motors, Car Chassis, Motor Driver, Battery, Powerbank, Raspberry Pi |
| Priority | Low |
| Estimated Time Needed | 8 Hours |
| Dependency | Fully charged powerbank. |
| Setup | Give power to the Raspberry Pi. |
| Procedure | 1. Open the Raspberry Pi. 2. Check the time |
| Cleanup | Power off the Raspberry Pi. |

### 4.3.3. Manual Movement

|  |  |
| --- | --- |
| **TC\_ID** | **MMV.01** |
| Purpose | The Sentinel is manually moved in real life. |
| Requirements | Joystick (or Keyboard), Hardware Components. |
| Priority | High |
| Estimated Time Needed | 5-10 seconds |
| Dependency | The Hardware Components should be fully working. |
| Setup | Activate the motors using the motor driver and supply energy from battery, connect the Raspberry Pi, and connect a joystick or keyboard. |
| Procedure | 1. Activate Motors. 2. Connect the Raspberry Pi. 3. Run the movement code. 4. Control the joystick or keyboard from the Remote Computer. 5. Move around real life and explore the environment. |
| Cleanup | Close the Sentinel. |

|  |  |
| --- | --- |
| **TC\_ID** | **MMV.02** |
| Purpose | The Sentinel is manually moved in simulation. |
| Requirements | Joystick (or Keyboard), Car Model, ROS Visualization Apps |
| Priority | Medium |
| Estimated Time Needed | 15-20 seconds |
| Dependency | Car Model with the real car features. |
| Setup | Enable the Car model, and connect the joystick or keyboard. Activate visulization app. |
| Procedure | 1. Open the ROS Visualization App. (RViz, Gazebo) 2. Add the Car Model. 3. Connect the joystick or keyboard to the computer. 4. Run the Movement code for simulation. 5. Move around the simulation world. |
| Cleanup | Close the simulation. |

### 4.3.4. Autonomous Movement

|  |  |
| --- | --- |
| **TC\_ID** | **AMV.01** |
| Purpose | The Sentinel is autonomously moved in real life. |
| Requirements | Lidar, Hardware Components, Exploring Algorithms |
| Priority | Medium |
| Estimated Time Needed | 1-2 minutes. |
| Dependency | The Hardware Components and Exploring and Deciding algorithms should be fully working. |
| Setup | Activate the motors using the motor driver and supply energy from battery, connect the Raspberry Pi, and run the exploring and deciding path algorithms. |
| Procedure | 1. Activate Motors. 2. Connect the Raspberry Pi. 3. Run the autonomous movement codes. 4. Wait the explore the area, and decide the path to move. 5. After exploring the whole room area, stop the movement. |
| Cleanup | Close the Sentinel. |

|  |  |
| --- | --- |
| **TC\_ID** | **AMV.02** |
| Purpose | The Sentinel is autonomously moved in simulation. |
| Requirements | Lidar, Exploring Algorithms, Car Model, ROS Visualization Apps. |
| Priority | Medium |
| Estimated Time Needed | 1-2 minutes. |
| Dependency | The Car Model should be added to the ROS Visualization App, and the Exploring and Deciding algorithms should be fully working. |
| Setup | Enable the Car Model in the ROS Visualization App, and run the exploration and decision path algorithms. |
| Procedure | 1. Open the ROS Visualization App. (Gazebo) 2. Add the Car Model. 3. Run the autonomous movement codes. 4. Exploring the simulation world. 5. Move around the simulation world. |
| Cleanup | Close the simulation. |

### 4.3.5. Manual Mapping

|  |  |
| --- | --- |
| **TC\_ID** | **MM.2D.01** |
| Purpose | 2D mapping a single line |
| Requirements | Joystick, Lidar |
| Priority | Low |
| Estimated Time Needed | 5-10 seconds |
| Dependency | Accurate lidar data |
| Setup | Active Lidar sensor, connect joystick |
| Procedure | 1. Open simulation world 2. Clear all the objects 3. Run the movement code 4. Move the Sentinel forward and backward |
| Cleanup | Close the simulation |

|  |  |
| --- | --- |
| **TC\_ID** | **MM.2D.02** |
| Purpose | 2D mapping an empty virtual room |
| Requirements | Joystick, Lidar |
| Priority | Medium |
| Estimated Time Needed | 60-120 seconds |
| Dependency | Accurate lidar data |
| Setup | Active Lidar sensor, connect joystick |
| Procedure | 1. Open simulation world 2. Clear all the objects 3. Run the movement code 4. Move the Sentinel in four directions |
| Cleanup | Close the simulation |

|  |  |
| --- | --- |
| **TC\_ID** | **MM.2D.03** |
| Purpose | 2D mapping an object over a line |
| Requirements | Joystick, Lidar |
| Priority | Medium |
| Estimated Time Needed | 5-10 seconds |
| Dependency | Accurate lidar data |
| Setup | Active Lidar sensor, connect joystick |
| Procedure | 1. Open simulation world 2. Place one object to world 3. Run the movement code 4. Move the Sentinel forward and backward 5. Check the map for an object appearance from Lidar data |
| Cleanup | Close the simulation |

|  |  |
| --- | --- |
| **TC\_ID** | **MM.2D.04** |
| Purpose | 2D mapping a virtual room with many objects |
| Requirements | Joystick, Lidar |
| Priority | High |
| Estimated Time Needed | 3 minutes |
| Dependency | Accurate lidar data |
| Setup | Active Lidar sensor, connect joystick |
| Procedure | 1. Open simulation world 2. Place objects to different locations 3. Run the movement code 4. Move the Sentinel to recognize objects and the world 5. Check the created 2D map with respect to created world |
| Cleanup | Close the simulation |

|  |  |
| --- | --- |
| **TC\_ID** | **MM.2D.05** |
| Purpose | 2D mapping in a real room |
| Requirements | Joystick, Lidar |
| Priority | High |
| Estimated Time Needed | 4 minutes |
| Dependency | Accurate lidar data |
| Setup | Active Lidar sensor, connect joystick, activate motors |
| Procedure | 1. Connect raspberry pi to power 2. Open motors’ switch 3. Run the movement code from Sentinel 4. Run the movement code from Server 5. Launch Rviz2 and SlamToolBox 6. Move the Sentinel inside the room 7. Check if 2D map matches with real room |
| Cleanup | Close the Sentinel and other code blocks |

|  |  |
| --- | --- |
| **TC\_ID** | **MM.3D.01** |
| Purpose | Create 3D map of simulated world |
| Requirements | Joystick, Lidar, Camera |
| Priority | Medium |
| Estimated Time Needed | 3 minutes |
| Dependency | Accurate lidar data and camera data |
| Setup | Active Lidar sensor, activate camera, connect joystick |
| Procedure | 1. Open simulation world 2. Place the objects into desired locations 3. Run the movement code 4. Move the Sentinel to recognize objects and the world 5. Put images and lidar data together 6. Create 3D map using Rtabmap 7. Check if the generated map is matches the simulated world |
| Cleanup | Close the simulation |

|  |  |
| --- | --- |
| **TC\_ID** | **MM.3D.02** |
| Purpose | Create 3D map of real life room |
| Requirements | Joystick, Lidar, Camera |
| Priority | High |
| Estimated Time Needed | 4 minutes |
| Dependency | Accurate lidar data and camera data |
| Setup | Active Lidar sensor, activate camera , connect joystick, activate motors |
| Procedure | 1. Connect raspberry pi to power 2. Open motors’ switch 3. Run the movement code from Sentinel 4. Run the movement code from Server 5. Launch Rtabmap 6. Move Sentinel to explore the room 7. Put images and lidar data together 8. Create 3D map using Rtabmap 9. Check if the generated map is matches the simulated world |
| Cleanup | Close the Sentinel and code blocks |

### 4.3.6. Autonomous Mapping

|  |  |
| --- | --- |
| **TC\_ID** | **AM.2D.01** |
| Purpose | 2D mapping in simulation autonomously |
| Requirements | Joystick, Lidar |
| Priority | High |
| Estimated Time Needed | 3 minutes |
| Dependency | Accurate lidar data |
| Setup | Active Lidar sensor, connect joystick |
| Procedure | 1. Open simulation world 2. Place the objects into desired locations 3. Run Rviz2 4. Run the autonomous movement code 5. Wait for the Sentinel to move around and recognize objects 6. Check if the generated 2D map is matches the simulated world 7. Check for possible crashes |
| Cleanup | Close the Simulation and other code blocks |

|  |  |
| --- | --- |
| **TC\_ID** | **AM.2D.02** |
| Purpose | 2D mapping in a real room autonomously |
| Requirements | Joystick, Lidar |
| Priority | High |
| Estimated Time Needed | 4 minutes |
| Dependency | Accurate lidar data |
| Setup | Active Lidar sensor, connect joystick, activate motors |
| Procedure | 1. Connect raspberry pi to power 2. Open motors’ switch 3. Run the autonomous movement code from Sentinel 4. Run the autonomous movement code from Server 5. Run Rviz2 6. Wait for the Sentinel to move around and recognize objects 7. Check if the generated 2D map is matches the simulated world 8. Check for possible crashes |
| Cleanup | Close the Sentinel and other code blocks |

|  |  |
| --- | --- |
| **TC\_ID** | **AM.3D.01** |
| Purpose | 3D mapping in a simulation autonomously |
| Requirements | Joystick, Lidar, Camera |
| Priority | High |
| Estimated Time Needed | 3 minutes |
| Dependency | Accurate lidar data and camera data |
| Setup | Active Lidar sensor, activate camera , connect joystick |
| Procedure | 1. Open simulation world 2. Place the objects into desired locations 3. Run Rviz2, Rtabmap 4. Run the autonomous movement code 5. Wait for the Sentinel to move around and recognize objects 6. Check if the generated 3D map is matches the simulated world |
| Cleanup | Close the simulation and other code blocks |

|  |  |
| --- | --- |
| **TC\_ID** | **AM.3D.02** |
| Purpose | 3D Mapping in real room autonomously |
| Requirements | Joystick, Lidar,Camera |
| Priority | High |
| Estimated Time Needed | 4 minutes |
| Dependency | Accurate lidar data and camera data |
| Setup | Active Lidar sensor, activate camera , connect joystick, activate motors |
| Procedure | 1. Connect raspberry pi to power 2. Open motors’ switch 3. Run the autonomous movement code from Sentinel 4. Run the autonomous movement code from Server 5. Run Rviz2, Rtabmap 6. Wait for the Sentinel to move around and recognize objects 7. Check if the generated 3D map is matches the simulated world |
| Cleanup | Close the Sentinel and other code blocks |

## 

### 4.3.7. Object Detection

|  |  |
| --- | --- |
| **TC\_ID** | **OD.01** |
| Purpose | Detect the objects in the simulation world. |
| Requirements | Camera, Movement, YOLO model. |
| Priority | Medium |
| Estimated Time Needed | 3 minutes |
| Dependency | Camera data |
| Setup | Active camera, download object detection model. |
| Procedure | 1. Open simulation world. 2. Listen camera and start object detection model. 3. Place the objects into the desired location in the simulation world. 4. Turn Sentinel’s camera to the object. 5. Wait model to detect objects. 6. Check the detected object. |
| Cleanup | Close the Sentinel and other code blocks |

|  |  |
| --- | --- |
| **TC\_ID** | **OD.02** |
| Purpose | Detect the objects in the real world. |
| Requirements | Camera, Movement, YOLO model. |
| Priority | Medium |
| Estimated Time Needed | 15 minutes |
| Dependency | Camera data |
| Setup | Active camera, download object detection model. |
| Procedure | 1. Connect Raspberry Pi to power 2. Open motors’ switch 3. Run the movement and camera code from Sentinel 4. Run the movement code from Server 5. Listen camera and start object detection model 6. Turn Sentinel’s camera to the object 7. Wait model to detect objects. 8. Check the detected object |
| Cleanup | Close the Sentinel and other code blocks |

|  |  |
| --- | --- |
| **TC\_ID** | **OD.03** |
| Purpose | Distinguish multiple object from each other. |
| Requirements | Camera, Movement, YOLO model. |
| Priority | Medium |
| Estimated Time Needed | 15 minutes |
| Dependency | Camera data |
| Setup | Active camera, download object detection model. |
| Procedure | 1. Open Simulation or Start Sentinel. 2. Listen camera and start object detection model. 3. Place the different objects in the desired locations with close distances. 4. Turn Sentinel’s camera to the objects. 5. Wait model to detect objects. 6. Check each object detected properly. |
| Cleanup | Close the Sentinel and other code blocks |

|  |  |
| --- | --- |
| **TC\_ID** | **OD.04** |
| Purpose | Detect objects from different distances. |
| Requirements | Camera, Movement, YOLO model. |
| Priority | Medium |
| Estimated Time Needed | 8 minutes |
| Dependency | Camera data |
| Setup | Active camera, download object detection model. |
| Procedure | 1. Open Simulation or Start Sentinel 2. Listen camera and start object detection model 3. Place objects at multiple known ranges like close, mid-range or far. 4. Turn Sentinel’s camera to the objects 5. Wait model to detect objects. 6. Check the detected object for each distance. |
| Cleanup | Close the Sentinel and other code blocks |

|  |  |
| --- | --- |
| **TC\_ID** | **OD.05** |
| Purpose | Evaluate the reliability of detection by evaluating the rate of false positives and negatives. |
| Requirements | Camera, Movement, YOLO model. |
| Priority | Medium |
| Estimated Time Needed | 20 minutes |
| Dependency | Camera |
| Setup | Active camera, download object detection model. |
| Procedure | 1. Open Simulation or Start Sentinel 2. Listen camera and start object detection model 3. Place objects in a patterned background or with reflections. 4. Turn Sentinel’s camera to the objects. 5. Wait model to detect objects. 6. Log instances of false detection and missed detections. 7. Evaluate the model performance. |
| Cleanup | Close the Sentinel and other code blocks |

|  |  |
| --- | --- |
| **TC\_ID** | **OD.06** |
| Purpose | Ensure the detection algorithm meets real-time performance requirements during both manual and autonomous operation. |
| Requirements | Camera, Movement, YOLO model. |
| Priority | Medium |
| Estimated Time Needed | 1 hour |
| Dependency | Camera |
| Setup | Active camera, download object detection model. |
| Procedure | 1. Open Simulation or Start Sentinel 2. Listen camera and start object detection model 3. Place the different objects in the desired and distinct locations. 4. Move Sentinel around the environment. 5. Monitor the latency between object appearance and detection reporting. |
| Cleanup | Close the Sentinel and other code blocks |

### 4.3.8 Simulation

|  |  |
| --- | --- |
| **TC\_ID** | **S.01** |
| Purpose | Movement of the Sentinel on simulation world |
| Requirements | Remote Computer, ROS2 Jazzy, Gazebo Harmonic |
| Priority | High |
| Estimated Time Needed | 10 Minutes |
| Dependency | Installed RViz and Gazebo Harmonic |
| Setup | The accurate URDF model of Sentinel |
| Procedure | 1. Run simulation package 2. Compare the movement of Sentinel on RViz and Gazebo it must be same |
| Cleanup | Interrupt the package |

|  |  |
| --- | --- |
| **TC\_ID** | **S.02** |
| Purpose | 2D mapping |
| Requirements | Remote Computer, ROS2 Jazzy, Gazebo Harmonic |
| Priority | High |
| Estimated Time Needed | 10 Minutes |
| Dependency | Installed RViz, Gazebo Harmonic and SLAM Toolbox |
| Setup | The accurate URDF model of Sentinel |
| Procedure | 1. Run the simulation package 2. Start to mapping 3. Check the created map on RViz |
| Cleanup | Interrupt the package |

|  |  |
| --- | --- |
| **TC\_ID** | **S.03** |
| Purpose | 3D mapping |
| Requirements | Remote Computer, ROS2 Jazzy, Gazebo Harmonic |
| Priority | High |
| Estimated Time Needed | 10 Minutes |
| Dependency | Installed RViz and Gazebo Harmonic |
| Setup | The accurate URDF model of Sentinel |
| Procedure | 1. Run simulation package 2. Start to mapping 3. Check the created map on RTAB-Map |
| Cleanup | Interrupt the package |

|  |  |
| --- | --- |
| **TC\_ID** | **S.04** |
| Purpose | Object Avoidance |
| Requirements | Remote Computer, ROS2 Jazzy, Gazebo Harmonic |
| Priority | High |
| Estimated Time Needed | 10 Minutes |
| Dependency | Installed RViz and Gazebo Harmonic |
| Setup | The accurate URDF model of Sentinel |
| Procedure | 1. Run the simulation package 2. Put obstacle objects on the Gazebo Simulation World 3. Start autonomous movement 4. Check the movement |
| Cleanup | Interrupt the package |

|  |  |
| --- | --- |
| **TC\_ID** | **S.05** |
| Purpose | Movement of the Sentinel on simulation world |
| Requirements | Remote Computer, ROS2 Jazzy, Gazebo Harmonic |
| Priority | High |
| Estimated Time Needed | 10 Minutes |
| Dependency | Installed RViz and Gazebo Harmonic |
| Setup | The accurate URDF model of Sentinel |
| Procedure | 1. Run the simulation package 2. Publish a 2D pose estimation on RViz 3. Check the movement |
| Cleanup | Interrupt the package |

## 4.4. Features To Be Not Tested

### 4.4.1. Windows Operating System Compatibility

Since all the developments and validations are performed on Ubuntu systems, testing on Windows environment are excluded. This prevents inconsistencies result from operating system-specific behaviors that are out of scope.

### 4.4.2. Driving on Flat Surfaces Only

All the test are limited to flat surfaces, since the Sentinel’s navigation and performance are only validated in these conditions. Testing on the rough terrain is beyond of the current scope of the Sentinel.

### 4.4.3. Well-Lit Environment

Testings are performed in a controlled, well-lit environment to ensure consistent sensor performance and reduce variability. Low or variable lighting conditions are outside the current test parameters.

### 4.4.4. Limited to the ROS-Jazzy and Gazebo Harmonic Versions

The system only be evaluated using ROS-Jazzy and Gazebo Harmonic versions. Compatibility with older versions or alternative releases will not be tested.

### 4.4.5. Internet Connectivity and Real Time Data

The Sentinel is designed to remain constantly connected to the internet to send and receive real-time data for remote monitoring, control, and updates. Testing without an internet connection or under intermittent network conditions is out of scope for the Sentinel.

## 4.5. Pass/Fail Criteria

### 4.5.1. Manual Movement

The Sentinel must move in four directions without struggling. Also, the linear speed of the Sentinel must be adjustable.

### 4.5.2. Autonomous Movement

The Sentinel will use an autonomous movement algorithm that will handle driving without any crashes and explore the whole room that Sentinel has in it.

### 4.5.3. Manual Mapping

The algorithm should create at least 95% accurate 2D maps in a simulation environment and 90% accurate 2D maps in real life conditions while the Sentinel is driven by the user.

The 3D map creating algorithm should properly create objects in the correct location of themselves and place them accordingly with a maximum of 10% error margin in both simulation and real life conditions while the Sentinel is driven by the user.

### 4.5.4. Autonomous Mapping

The algorithm should create at least 95% accurate 2D maps in a simulation environment and 90% accurate 2D maps in real life conditions while the Sentinel is driven by the autonomous drive algorithm.

The 3D map creating algorithm should properly create objects in the correct location of themselves and place them accordingly with a maximum of 10% error margin in both simulation and real life conditions while the Sentinel is driven by the autonomous drive algorithm.

### 4.5.5. Object Detection

The object detection algorithm must process frames fast enough to catch the camera stream, and must classify objects with respect to these images. The accuracy of classifying objects should have accuracy higher than 75%.

### 4.5.6. Simulation

The simulation must mimic real world scenarios such as friction, objects, lidar data and camera view. Simulation and real-world movement must be identical to each other. (For example, if Sentinel turns with 45 degrees angle, the simulated car must also be turned with 45 degrees angle.

## 4.6. Exit Criteria

The testing of the Sentinel is considered successful under the following conditions:

* 100% of the test cases are executed.
* 90% of the test cases passed.
* All High and Medium Priority test cases passed.

## 4.7. Test Results

|  |  |  |  |
| --- | --- | --- | --- |
| 1. TC\_ID | Priority | Result | Explanation |
| WUI.01 | Low | Pass | Make movement visible on web browser. |
| WUI.02 | Low | Fail | User cannot control using keyboard. |
| WUI.03 | Low | Pass | User can see real-time video captured by camera. |
| WUI.04 | Low | Pass | Make visible 2D map on web browser. |
| WUI.05 | Low | Pass | Make visible 3D Car model with movement direction. |
| WUI.06 | Low | Pass | Make visible 3D Map on web browser. |
| WUI.07 | Low | Pass | User can see logs. |
| H.01 | Low | Pass | Make controllable without any loose contact in cables. |
| H.02 | Low | Pass | Motor driver can deliver required voltage. |
| H.03 | Low | Pass | Motors can be turned (forward and backward). |
| H.04 | Low | Pass | Battery can deliver 1.5 hours of continuous operation. |
| H.05 | Low | Pass | Powerbank can deliver 8 hours of continuous operation |
| MMV.01 | High | Pass | Manual Movement can process while controlled by the user in real life. |
| MMV.02 | Medium | Pass | Manual Movement can process while controlled by the user in simulation. |
| AMV.01 | Medium | Pass | Autonomous Movement can process while Sentinel is moving without being controlled by the user in real life. |
| AMV.02 | Medium | Pass | Autonomous Movement can be processed while Sentinel is moving without being controlled by the user in simulation. |
| MM.2D.01 | Low | Pass | The 2D map can be created while Sentinel is moving with user control in simulation. |
| MM.2D.02 | Medium | Pass | The 2D map can be created while Sentinel is moving with user control in different directions in simulation. |
| MM.2D.03 | Medium | Pass | The 2D map can be created while Sentinel is moving with user control in a straight line in simulation. |
| MM.2D.04 | High | Pass | The 2D map can be created while Sentinel is moving around with user control with different objects in simulation. |
| MM.2D.05 | High | Pass | The 2D map can be created while Sentinel is moving around with user control with different objects in the real room. |
| MM.3D.01 | Medium | Pass | The 3D map can be created while Sentinel is moving around with user control with different objects in simulation. |
| MM.3D.02 | Low | Fail | The 3D map cannot create in the real room. |
| AM.2D.01 | High | Pass | The 2D map can be created while Sentinel is moving, using an autonomous movement algorithm in a simulation. |
| AM.2D.02 | High | Pass | The 2D map can be created while Sentinel is moving, using an autonomous movement algorithm in a real room. |
| AM.3D.01 | High | Pass | The 3D map can be created while Sentinel is moving, using an autonomous movement algorithm in a simulation. |
| AM.3D.02 | Low | Fail | The 3D map cannot create while Sentinel is moving, in a real room. |
| OD.01 | Medium | Pass | Object detection algorithms can actively perform to identify and detect objects in the simulation world. |
| OD.02 | Medium | Pass | Object detection algorithms can actively perform to identify and detect objects in the real world. |
| OD.03 | Medium | Pass | The system can identify and distinguish multiple objects in an environment. |
| OD.04 | Medium | Pass | The system can be detected objects at varying distances, adjusting its focus to accurately identify them. |
| OD.05 | Medium | Pass | Evaluate the accuracy of detected objects. |
| OD.06 | Medium | Pass | Object detection algorithm meets real-time performance requirements during movement. |
| S.01 | High | Pass | The movement of the Sentinel is the same on RViz and Gazebo. |
| S.02 | High | Pass | The Sentinel can be created a 2D map of the Gazebo simulation world. |
| S.03 | High | Pass | The Sentinel can be created a 3D map of the Gazebo simulation world. |
| S.04 | High | Pass | The Sentinel can move by avoiding objects on the Gazebo simulation world. |
| S.05 | High | Pass | The Sentinel can move autonomously to a published point on the Gazebo simulation world. |

## 4.8. Summary of the Test Results

According to our test results, we provided 3 of the Exit Criteria:

* 100% of the test cases are successfully executed.
* 90% of the test cases successfully passed. (We have 35 pass 3 fail test cases. %92 of the test cases passed.)
* All High and Medium Priority test cases passed.

# References

|  |  |
| --- | --- |
| [1] | M. Labbe, “rtabmap\_ros,” 10 December 2023. [Online]. Available: http://wiki.ros.org/rtabmap\_ros. [Accessed 1 December 2024]. |
| [2] | “YOLO: Real-Time Object Detection,” [Online]. Available: https://pjreddie.com/darknet/yolo/. [Accessed 1 December 2024]. |
| [3] | “rosbag,” [Online]. Available: https://wiki.ros.org/rosbag. [Accessed 1 December 2024]. |
| [4] | 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]. |