OULUN AMMATTIKORKEAKOULU

Aki Panuma

Building a Web-Based User Interface for Mobile Robots

Building a Web-Based User Interface for Mobile Robots

Aki Panuma
Thesis
Autumn 2023
Bachelor of Engineering, Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Bachelor of Engineering, Information Technology

Author(s): Aki Panuma
Title of the thesis: Building a Web-based User Interface for Mobile Robots
Thesis examiner(s): Jouni Juntunen
Term and year of thesis completion: Autumn 2023 Pages: 29

The theme of this thesis was to provide a proof-of-concept web-based user interface for mobile robots, focusing on the integration of 3D graphics using Three.js within a web environment. Aimed at enhancing the control and interaction with mobile robots, the project was developed for Probot Oy, a robotics and automation company based in Oulu.

The document covers the background, objectives, methodology, and the implementation process, including planning, development, and user interaction features. It highlights the significance of 3D graphics in mobile robotics user interfaces, demonstrating through practical examples how Three.js can be effectively used to create a three-dimensional mobile robot user interface and whether Three.js is suitable for such a task. The thesis concludes with reflections on the project's success, insights gained, and future opportunities for expansion and improvement.

The project was successful, and the company gained important insight into Three.js. Producing 3D graphics and setting the scene was made easy by the vast development documentations and practical examples. The company did not have previous experience with Three.js but when comparing technology options for this project, it was deemed the most suitable based on the available documents and the compatibility with web development guidelines and importing of 3D models.

Keywords: Robots, User Interfaces, Robotics, Three.js, 3D

CONTENTS

1 INTRODUCTION

In the current world of mobile robotics and automation, the need for scalable, intuitive, and easy to use interfaces has never been more pressing. As automation and robotics become more accessible to companies and the public alike, the ways in which humans interact with these machines plays a crucial role in the effectiveness and adaptation of new technologies. This thesis will focus on the technical implementation of Three.js as a web-based user interface front-end, which is visually appealing, easy to use and scalable with the intent of providing a bridge between robotic functionality and user-friendly interactions. Insights gained during this project will be shared with the company to enhance following iterations.

1.1 Objectives

The task is to build a mobile robot user interface as a POC (proof-of-concept) web application. The application is planned to be used to control a mobile robot Dolly™, created by Probot Oy. The robot is intended to work as a mobile platform that provides solutions to logistical challenges in factories and facilities. For example, a robot arm with a gripper can be attached to it to remotely pick up items and complete tasks. Dolly™ can be controlled manually by a joystick, however, to completely benefit from the mobile nature of the robot and to automate the movements of the robot, a graphical user interface is needed to control the navigation, movement paths and current tasks.

The main objective of this thesis is to analyse the creation of 3D graphics in web-based user interfaces. In the domain of mobile robotics, the utility of 3D graphics within web-based user interfaces is becoming increasingly important, offering a transformative impact on control and interaction. Aucoin and Jenkin emphasize the scarcity of autonomous systems utilizing immersive 3D user interfaces, which suggests a significant opportunity for innovation in this space. (Aucoin, Nicole & Jenkin, Michael 2000.)

The finished application should work as a base for future mobile robot projects inside Probot Oy and even though the company does not have any previous project experience with Three.js, it was requested to be used with the UI for possible future virtual reality (VR) versions. In the case of the UI working as the visual side of the mobile robot, any functional requirements of it are excluded of

this thesis. Incorporating Three.js as the base for the project gives the option to further expanding the UI to the VR realm.

To ensure accessibility, cross-platform compatibility, and user-friendliness, the UI is required to be web-based. This thesis primarily emphasizes the desktop experience, and, as a POC, experiences on other devices are not of concern.

1.2 Mobile robots

Mobile robots are robots that can move in their environment. Their most typical use is for logistical purposes in industrial locations where repetitive and manual tasks can be automated. Mobile robots can be split into two main groups, Autonomous Mobile Robots (AMR) and Automated Guided Vehicles (AGV). AMRs do not rely on pre-set movement paths due to their nature of being able to map and navigate their surrounding with several sensors built into them. AGVs however require a programmed path that it follows using laser scanners, barcodes or magnetic strips embedded into the ground (Suomen Robotiikkayhdistys ry. 2023, 142). The Omron LD mobile robot, as depicted in Figure 1, showcases the capabilities of AMRs in an industrial setting.

Mobile robots are a fast-growing market segment but continuous upgrades and iterations within the segment are required to each of their functional components. Different mobile robot configurations require somewhat similar components, for example accurate MCUs (motor control units), environmental mapping for navigation and safety, connectability, artificial intelligence and real-time communications for IoT (Internet of Things) usage. (Suomen Robotiikkayhdistys ry. 2023, 142).

Within the last several years the largest percentual growth has happened in the hospitality industry. Service robots are divided into two main groups, industrial service robots and domestic service robots. Industrial service robots are often found in places that require logistics handling, for example hospitals and factories, where these robots move items and handle different repetitive tasks. These industrial robots are often large investments, and the market volume of industrial mobile robots is nearly four times higher than that of domestic service robots. Due to Covid-19, over 30 different disinfecting robots have appeared on the market in 2022 alone. Domestic mobile robots however perform tasks that are commonly found outside the industrial environments and within consumer homes. Due to the inexpensive vacuum cleaning and grass cutting robots, the sales of consumer-

grade domestic service robots crossed over ten million units sold yearly in the year 2020. (Suomen Robotiikkayhdistys ry 2023, 142-143.)

FIGURE 1. Omron LD- series mobile robot (Omron, 2023)

1.3 Graphical User Interfaces

User interfaces (UI) are the points of interaction between a user and a computer system, particularly the use of input devices and software (IxDF, 2016). They provide the means for users to control and operate the complex processes and machinery of a computer system. Graphical user interface (GUI) is a form of user interface which combines visible elements such as buttons and maps for navigating the functions of the platform which is to be controlled (Wikipedia, 2023).

Three-dimensional graphics user interfaces are often used in games and different visual environments. Many commonly known elements benefit of providing three dimensional visual elements on a screen for example Windows Aero and Aqua for MacOS to provide visual improvements and effects that would otherwise be difficult or resource intensive to produce in a 2D environment. (Wikipedia, 2023.)

In mobile robotics, graphical user interfaces are generally used to control movement paths and understanding the autonomous decisions made by mobile robots. GUIs constrain user inputs to

valid ranges and units, supply output where needed and provide users with sensory data required for controlling the mobile robot. (Jet Propulsion Laboratory, 2023.)

2 ABOUT 3D GRAPHICS

2.1 3D graphics

3D computer graphics represent geometric data in three dimensions for rendering primarily 2D digital images on displays. Despite their name, 3D graphics are often shown on 2D screens, providing an illusion of depth. The creation of 3D graphics involves several stages: modelling the shape of objects, animating, and placing them within a scene, and rendering, which transforms the model into an image through light simulation or artistic styling. The term “computer graphics” originates from the 1960s and was originally invented by William Fetter while working with Boeing. (Wikipedia, 2023.)

2.1.1 Scene

Every computer rendered object requires three important components, a scene, a source of light and a camera or an eye that views the scene (Threejs.org, 2023). Scenes in 3D- development work as containers for elements such as lighting, geometry, and cameras. Scene creation differs from environment to environment and while many applications create the scene automatically, environments such as Three.js require the manual creation of a scene.

Scenes work as a space for objects and animations. Animations involve placement and movements of objects within a 3D scene, setting their location, and size. Animation adds dimension to these objects, describing how they move and change over time. Techniques such as keyframing, inverse kinematics, and motion capture are employed to create dynamic movements within the scene. (Wikipedia, 2023.) After the scene has been set up and the geometry has been made visible with appropriate lighting, the scene can be rendered (Wikipedia, 2023).

2.1.2 Model

In the realm of 3D computer graphics, objects are commonly known as 3D models. 3D models are mathematical representations of objects, which become graphics upon rendering and displaying. These models can be created on a computer via CAD (Computer-aided design) (FIGURE 2) or

scanned from real-world objects. Rendering involves light transport and scattering calculations to produce realistic images. (Wikipedia, 2023.)

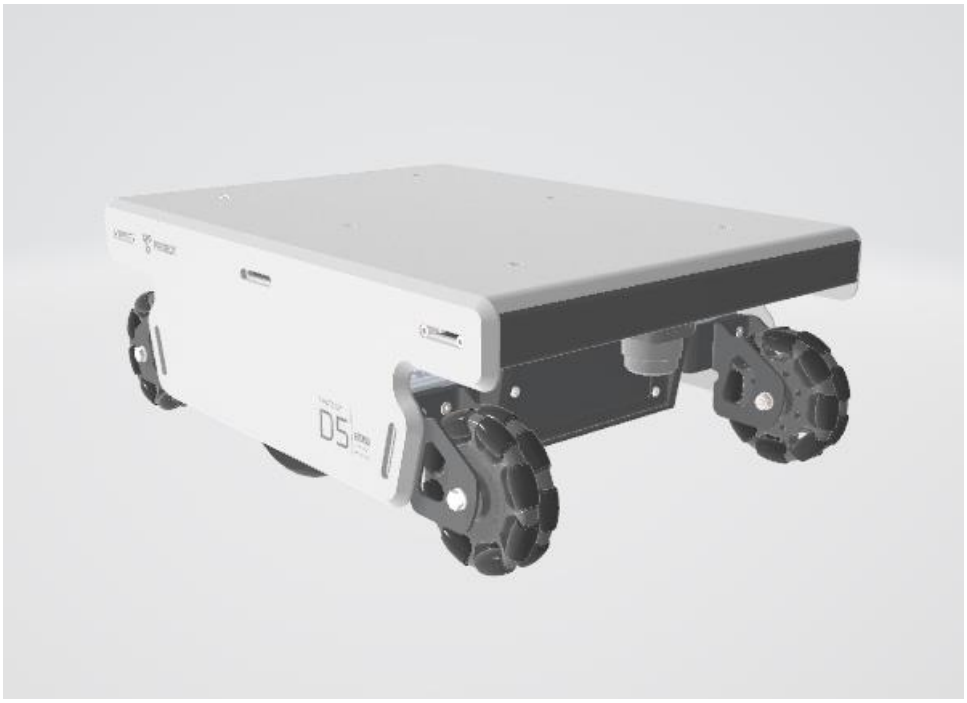

FIGURE 2. 3D- model of the mobile robot Dolly™ by Probot, created in SolidWorks CAD software, viewed in Windows 3D Viewer.

3D models can be created in various 3D graphics software, often called modelling applications. Almost all 3D models can be divided into two main categories, solid, and shell. While solid models represent the complete volume of the object they depict, shell or boundary models illustrate the surface or the boundary of an object rather than its volume. These models are hollow and most visual models in video games fall into this category due to the ease of rendering and performance benefits compared to solid models. Solid models are mostly utilized in engineering and simulations and are typically constructed using constructive solid geometry methods. (Wikipedia, 2023.)

CAD is often used for building 3D models and representations for industrial uses, for example a completed design can be brought to life by 3D printing it or by machining it from a solid material. CAD improves the overall quality of designs and streamlines the production due to the ease of sharing production documents. Models can be created from 2D sketches or curves, surfaces, and solids in the 3D space. These CAD designs can also be used as blueprints for industrial assemblies. (Wikipedia, 2023).

2.1.3 Textures

3D texturing is a process in computer graphics that adds realism and intricate details to digital models. By adding simple imperfections and blending different textures, the quality of an otherwise flat 3D model can be hugely improved. Textures are square bitmap images that are wrapped around 3D models adding to their realism. These images are designed to seamlessly repeat while applied on the model. In the animation industry, texturing and texture mapping play a crucial role. It is beneficial for giving surfaces proper looks ranging from shiny objects to fuzzy clothes. (MAP Systems, 2023.)

Texture mapping is a technique in 3D texturing used to apply textures such as detailed imagery, surface patterns, or colours to computer-generated models. It works by projecting a 2D image, (the texture), onto a 3D object. Coordinates from the object's surface are mapped to corresponding points on the texture, allowing the image to wrap around the object realistically. This process is refined using various techniques like UV unwrapping and filtering to ensure the texture adheres naturally to the object's shape and contours. (Wikipedia, 2023.)

UV unwrapping is a subprocess of 3D texturing where UV coordinates are generated for each vertex of a polygon mesh. The mesh is unfolded at the seams, and the triangles are laid out on a flat surface. Artists can then paint textures directly onto this unwrapped mesh. UV maps can be generated automatically or manually, with the option to adjust and optimize to minimize seams and overlaps. Essentially, UV unwrapping converts the complex 3D shape into a 2D representation that can be easily textured. (Wikipedia, 2023.)

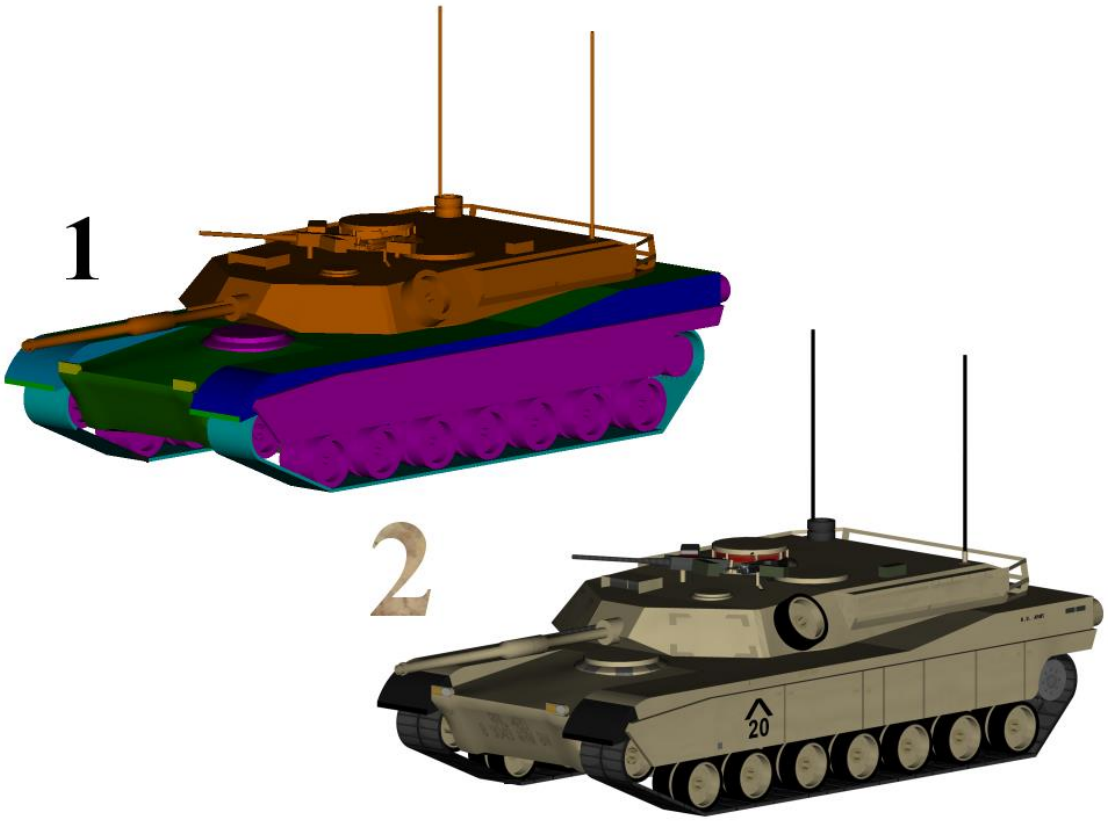

FIGURE 3. Textured (1.) vs non-textured (2.) 3D model of a M1A2 Abrams (Wikipedia, 2023).

2.1.4 Rendering

3D rendering is a process of 3D graphics for converting 3D models into 2D images on a computer. It includes photo realistic effects or non-photorealistic styles most used in fields like architecture, product design, photography, and virtual simulations. Rendering methods can be split into two which are real-time rendering and non-real-time rendering. (Wikipedia, 2023.)

Real-time rendering is most often used in video games and interactive graphics, where images must be computed at a high pace based on user interactions. The primary goal of real-time rendering is to produce as photorealistic result as possible at a framerate sufficient for the human eye to register as movement, usually over 24 frames per second (FPS). While real-time rendering is usually polygonal, specialized and task-specific techniques such as scanline rendering are often employed in real-time rendering to efficiently process and shade surfaces, ensuring that light spreads accurately across surfaces to provide an immersive interactive experience. Other effects like motion blur, lens flares and depth of field can be used to further improve the immersion (Wikipedia, 2023.)

Non-real-time rendering is utilized in scenarios where the focus is on high-quality image detail and accuracy and in situations where longer processing times are expected, such as in animations and visual effects for movies. Unlike real-time rendering, it focuses on achieving photorealism and complex visual effects, often using techniques such as ray tracing or radiosity, and can take significantly longer, anywhere from seconds to hours per frame to render depending on the complexity and target quality. The process for rendering is computationally taxing given the complexity of photorealistic renders, but due to the advancements in computer chip designs and production of highly efficient components, high degree of realism can be achieved while keeping hardware costs reasonable. (Wikipedia, 2023.)

2.2 WebGL

WebGL stands for Web Graphics Library, which is a widely used, cross-platform and royalty-free open web standard for displaying 3D graphics in the web via HTML canvas elements. It is supported by most modern browsers including Mozilla Firefox, Google Chrome, Microsoft Edge and Apple Safari and it's based on OpenGL. (The Khronos Group Inc, 2023.) WebGL distinguishes itself in digital experiences through its adeptness at seamlessly rendering complex 3D graphics on the web, thereby shaping the future of interactive digital content. It offers substantial benefits for both novice and experienced developers in the creation and deployment of 3D graphics. (Dom Favata, 2023.)

WebGL makes visuals look better and more detailed by using code to create shapes and using a client's graphics card (GPU) to show these shapes on an HTML canvas. This makes it different from other ways of showing graphics. Additionally, there are many WebGL tools like Playcanvas and Three.js that developers can use. These tools give developers a plethora of resources to make web apps that make the most of what WebGL can do. (Slava Podmurnyi, 2021.)

2.3 Three.js

Three.js is a 3D library designed to help developers easily deploy 3D graphics in a web environment. It works by providing a set of objects and functions that developers can use to define 3D scenes, camera views, lighting, materials, textures, and geometries. Three.js stands as one of the longest-standing and widely acclaimed libraries in its category. Being cross-browser by nature, the Three.js API (Application Programming Interface), facilitates the creation of animated 3D computer

graphics that can be reproduced across different web platforms. With its source code hosted on GitHub for collaborative and community-driven development, Three.js works as a cornerstone for web developers looking to implement interactive 3D elements in the web environment. Three.js is known for its well-documented resources, simple examples and vast community making the development relatively easy for developers who are new to 3D graphics programming. It can also be easily integrated with other web technologies like HTML, CSS, and JavaScript. (The Khronos Group Inc, 2023.)

Three main things are required to display graphics with Three.js: scene, camera, and renderer. As shown in Figure 4, The camera amongst other elements is handed over to the scene, and then the scene, along with its contents, is transferred to the renderer, which projects a portion of the 3D scene to the 2D canvas element using WebGL. (Thomas Sifferlen, 2023.)

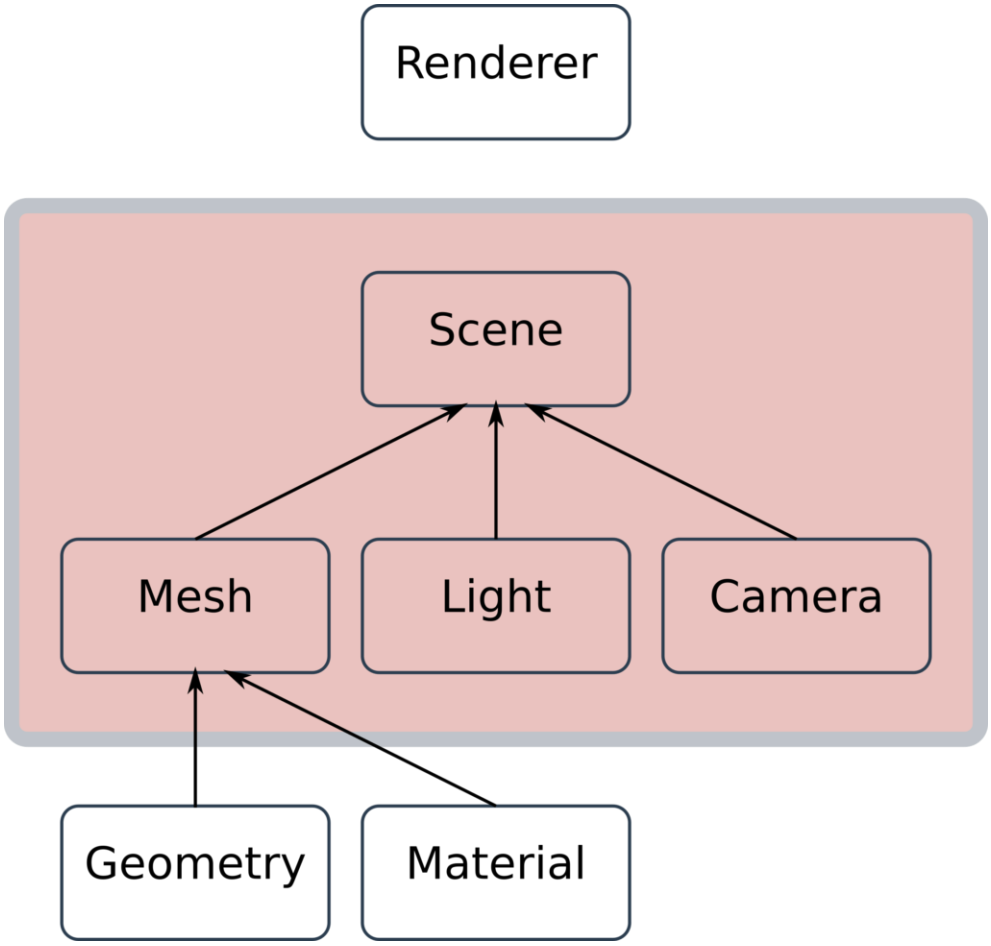

FIGURE 4. Three.js application structure diagram. (8)

Three.js brings the creation of 3D objects and scenes on web pages. Adding lights, shadows, and different materials improve the overall quality of the created objects. It also uses textures to add detailed designs to these shapes. The versatility of Three.js supports various applications, from interactive 3D experiences such as games and visualizations to practical tools for 3D artists. (Jawara Gordon, 2023.) These artists can utilize Three.js to create interactive galleries and portfolios, allowing real-time exploration of their work online. The library supports many standard 3D file formats including FBX, OBJ, gLTF and COLLADA (Threejs.org, 2023), making it a robust tool for professionals and hobbyists.

3 IMPLEMENTATION

This chapter dives into the practical realization of the user interface (UI) for a mobile robot, exploring the process from the initial plans to development. The implementation process is the base for the entire project and a crucial stage of creating a POC. Design choices are evaluated along with technologies that are used in the realization of the UI.

3.1 Planning

The initial drawings and design goals were derived from meetings with the CTO and CEO. The most critical parts of the UI were selected from these initial designs that were originally aimed exclusively for mobile phones. Complete freedom was given over the design and appearance of the user interface. The decision was made to focus on the implementation of functions within the navigation scene, which is the middle section of the initial UI sketch (FIGURE 5.).

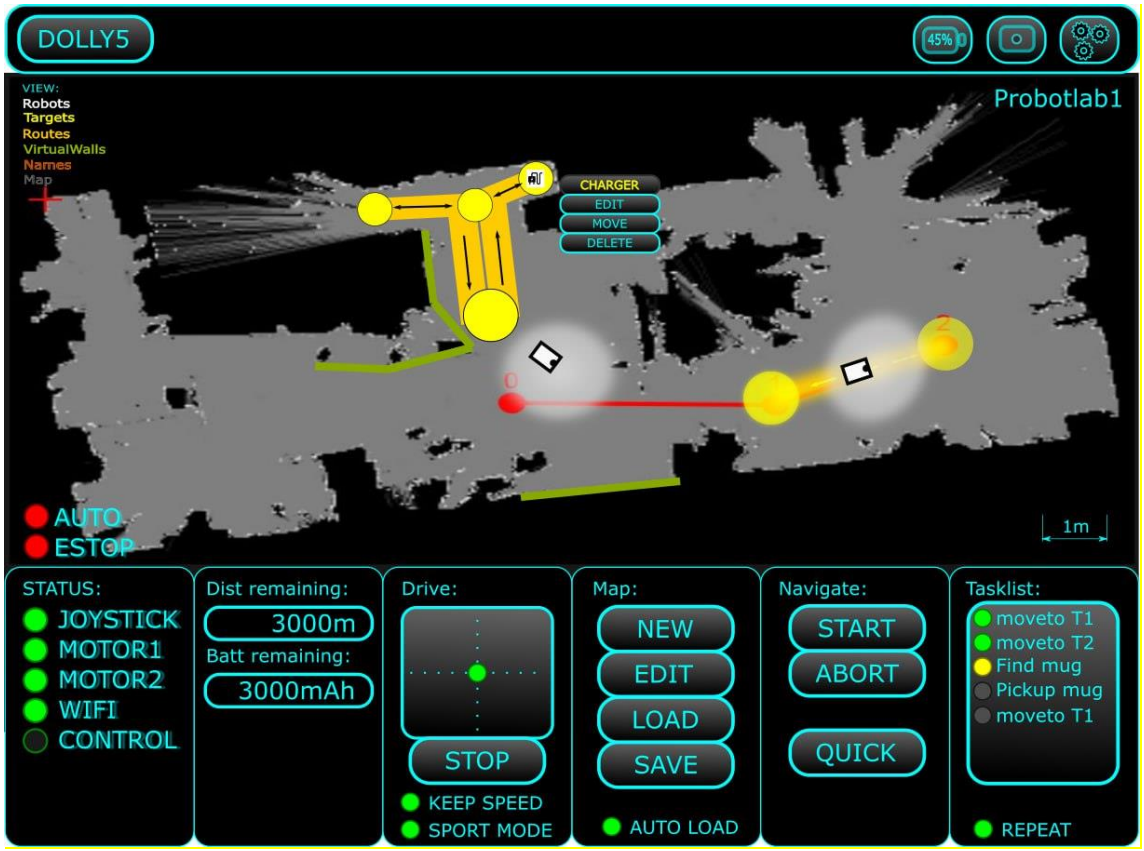

FIGURE 5. Initial pre-provided UI sketch.

The UI sketch has a light detection and ranging (Lidar) scan of an office that has been planted to the scene ground. Waypoints are located on top of the scan along with potential movement paths. The office scan is thought to be a part of the 3D UI and is meant to be inspected three dimensionally. The buttons laid out on the top and the bottom depict basic HTML elements.

3.2 Development results

The end creation is a POC, Three.js- based GUI with a few interactive elements for providing 3D-graphics in a web browser. It serves its initial purpose of testing out whether Three.js could be implemented into the form of a three-dimensional mobile robot UI. The GUI as it stands includes a scene, which works as the base for the Three.js environment, a renderer which projects the changes in the scene onto an HTML canvas and a camera that captures all this happening from the viewport and shows the scene from its perspective.

The scene was moulded around the technical limitations of Three.js and its basic functionalities with the programming style aimed to be as close to native JavaScript as possible. Several iterations of this UI were made which were then used for comparing different technologies and techniques. As a result of trying different VR compatible libraries and graphical utilities. The decision was made to stay completely within JavaScript and Three.js due to the ease of development.

3.3 Scene creation

The foundation of any Three.js application is the creation of a scene which serves as a space where all our 3D objects reside. To create a working Three.js environment, a camera and a renderer are also required. The scene is created by wrapping the Three.js code within HTML, along with the camera and renderer (FIGURE 6).

```html
<!DOCTYPE html>
<html>
<head>
    <title>Spinning Cube with Three.js</title>
    <style>
        body { margin: 0; }
        canvas { display: block; }
    </style>
</head>
<body>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/three.js/r128/three.min.js"></script>
    <script>
        // Scene setup
        var scene = new THREE.Scene();
        var camera = new THREE.PerspectiveCamera(75, window.innerWidth/window.innerHeight, 0.1, 1000);
        var renderer = new THREE.WebGLRenderer();
        renderer.setSize(window.innerWidth, window.innerHeight);
        document.body.appendChild(renderer.domElement);

        // Creating a cube
        var geometry = new THREE.BoxGeometry();
        var material = new THREE.MeshBasicMaterial({ color: 0x00ff00 });
        var cube = new THREE.Mesh(geometry, material);
        scene.add(cube);

        camera.position.z = 5;

        // Animation loop
        function animate() {
            requestAnimationFrame(animate);

            // Rotate the cube
            cube.rotation.x += 0.01;
            cube.rotation.y += 0.01;

            // Render the scene
            renderer.render(scene, camera);
        }

        animate();
    </script>
</body>
</html>
```

FIGURE 6. Example of the Three.js code wrapped in HTML.

In the project, the scene is initialized to represent the operational environment of the mobile robot. The project started by adding a grid floor in the scene coordinates 0, 0, -500. This floor location works as the base for the mobile robot map view. Scene creation in the form of a user interface was designed so that the views function by moving the camera within the scene and rendering the changes in a separate render loop.

3.4 Camera

The camera hovers over the created views while directly pointing at the centre of the views crafting the illusion of 2D elements in a 3D space (FIGURE 7). The camera must be placed somewhere in

the scene. Multiple cameras can be used simultaneously to benefit from several different views. Light is required for objects to be visible to the camera and therefore to the user.

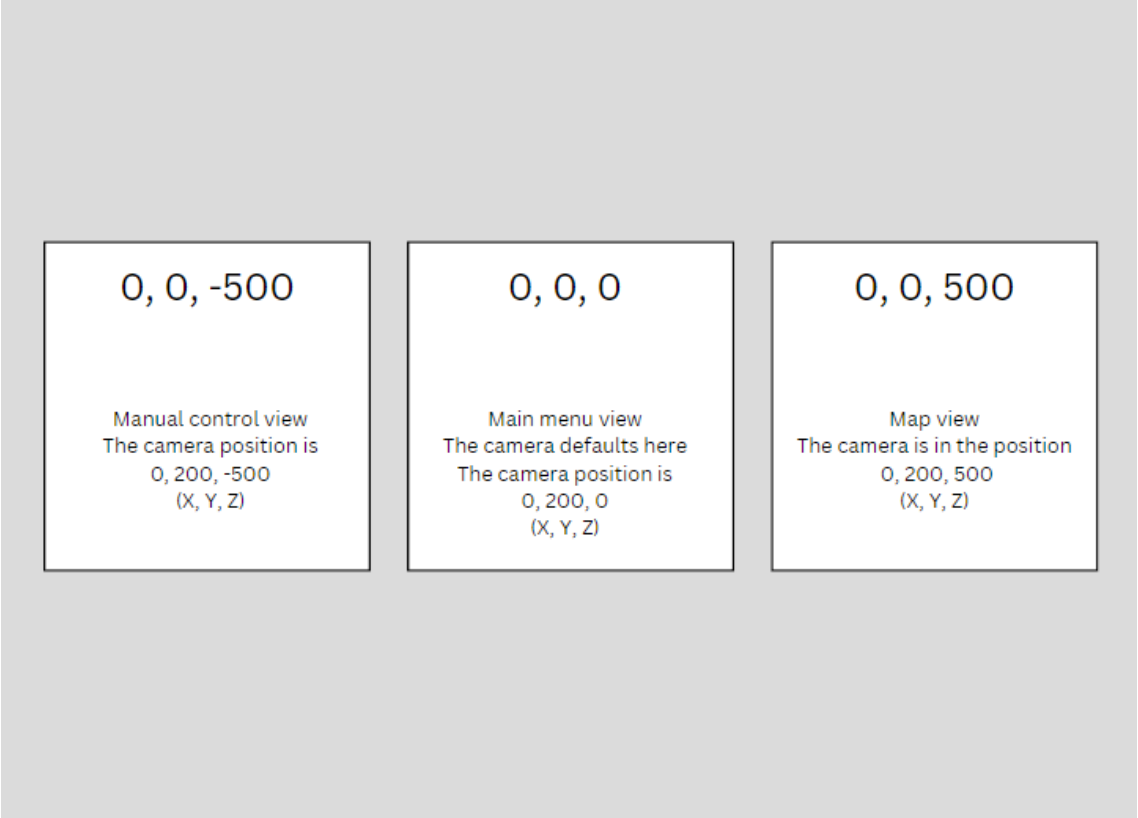

FIGURE 7. A photo representing the view locations within the Three.js scene. The number on top shows the location of the view in the scene. The Y- axis on the camera position shows the height of the camera from the floor of the view.

The main menu of the GUI is located at 0, 0, 0 within the scene coordinates. To view the 2D buttons placed at the centre of the scene, the camera must be moved up on the Y- axis and pointed downwards toward the button elements (FIGURE 8).

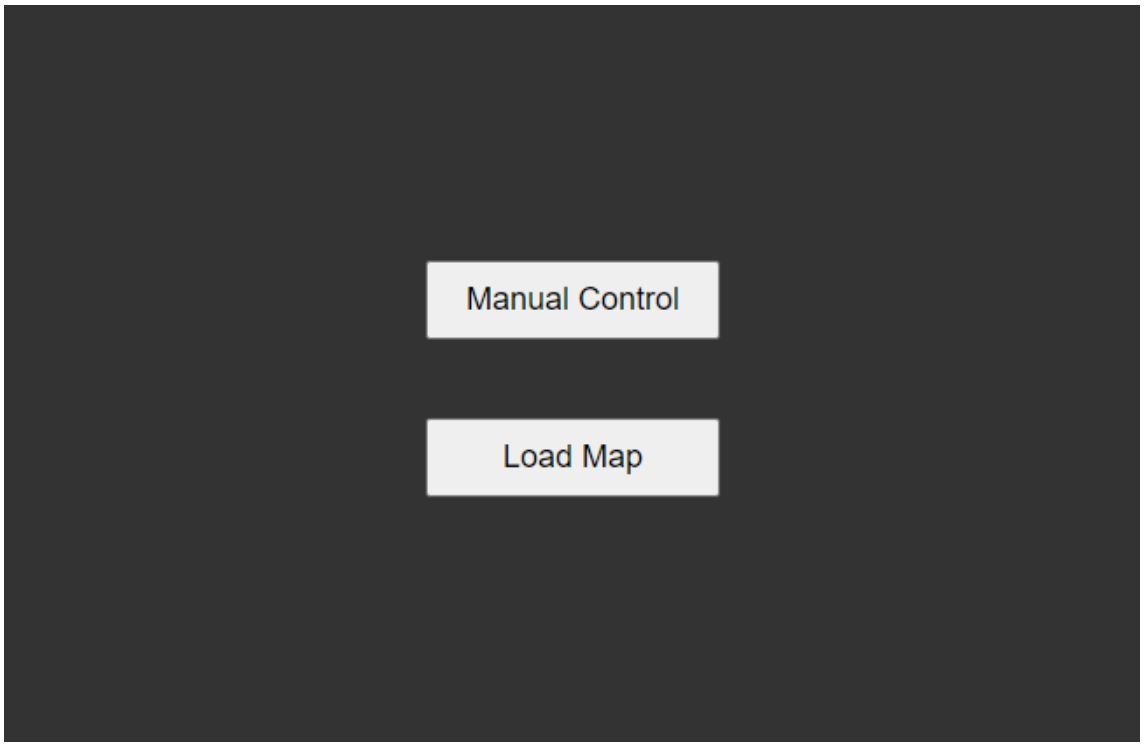

FIGURE 8. Main menu view. The camera is located 200 units above the scene pointing directly down.

It was essential to emphasize the 3D- element of the user interface which led to creating an instance of OrbitControls. This allowed the user to modify the view and move in the UI environment and around the grid if the camera were to be pointed at an incorrect place. This further allowed the UI to be viewed in proper 3D instead of the 2D view that the camera created while pointing straight down. OrbitControls is a Three.js addon that requires the dev to explicitly import it.

3.5 Rendering

Rendering in Three.js is the process of drawing the scene created by the developer onto the canvas element in the web page. After the scene creation, the renderer is called within an animation loop constantly updating the scene with any changes within the environment and redrawing the canvas to reflect user interactions in real-time. Every change that the developer wants to display must go through the render function. The renderer for this project was set up in the following way:

A renderer is created, and antialiasing is enabled. The size of the renderer is set to be the inner width and height of the container where the renderer is appended at the end of the snippet (FIGURE 9).

```
renderer = new THREE.WebGLRenderer({ antialias: true });
renderer.setPixelRatio(window.devicePixelRatio);
renderer.setSize(window.innerWidth, window.innerHeight);
renderer.shadowMap.enabled = true;
container.appendChild(renderer.domElement);
```

FIGURE 9. Setting up the renderer for this project and matching its size to the parent container.

For the application to constantly render the changes made into the scene, a renderer loop must be implemented. The loop uses “requestAnimationFrame”, which notifies the browser that the application wishes to perform an animation. The render loop is setup the following way:

```
function render() {

    requestAnimationFrame(render); // Schedule the next frame

    // Update logic here
    if (controls) {
        controls.update();
    }
    renderer.render(scene, camera); // Render the scene
}
```

FIGURE 10. Render loop for the UI. This function causes the screen to be rendered every time the screen is refreshed. On a traditional 60Hz display, the screen refreshes 60 times per second.

3.6 User interaction

One of the requirements for a working user interface is to have the required functionalities that a user can interact with the interface. Having built the UI as an initial test for possible adoption, some compromises were made in the functionalities of the UI.

The user interacts with the UI by setting waypoints in the environment which the robot navigates in. These waypoints are used as navigation points and are connected to other waypoints via navigation paths (FIGURE 11).

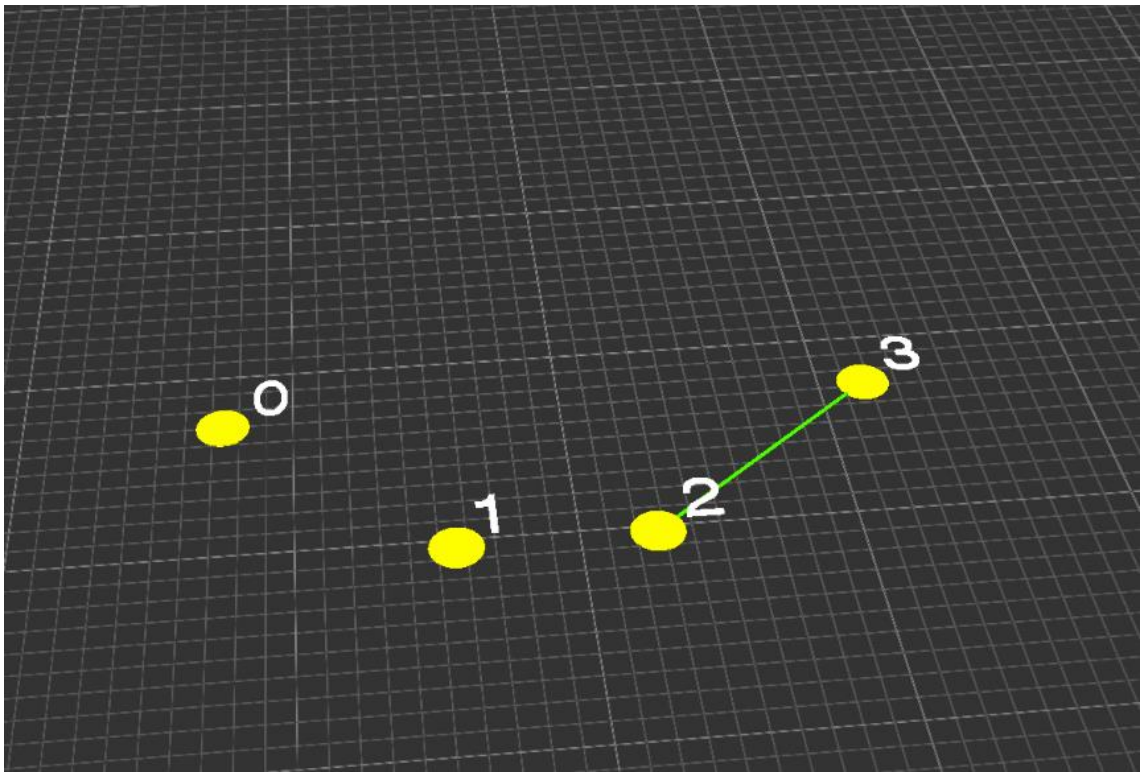

FIGURE 11. Waypoints and a navigation path in the scene.

The code gives each waypoint an ID on placement while also pushing the position, radius (r) and name to the waypoint mesh (FIGURE 12).

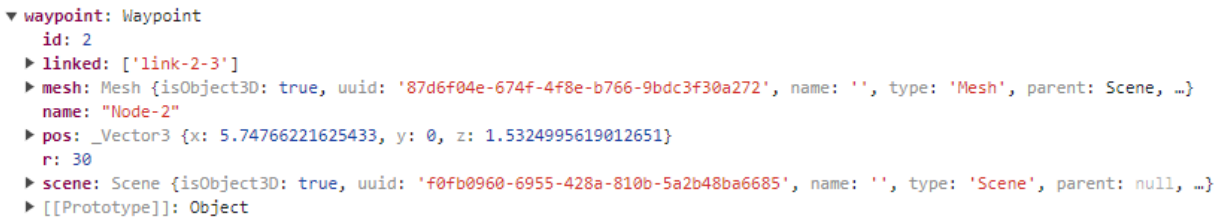

FIGURE 12. Construction of the waypoint mesh checked through Google Chrome developer console.

Real-Time feedback is being shown to the user in the form of an updating map. The mobile robot Dolly™ comes equipped with a Lidar, that scans the environment around the robot and notifies the user interface of changes in the environment and navigation. Lidar points are marked as blue boxes in the Three.js scene (FIGURE 13). The points marked by the Lidar are supposed to constantly update to depict the movement of the robot in its environment, but this function has not been implemented due to the POC stage of this UI. Most of the user interactions happen in the map view of the user interface, which as previously mentioned, is located at 0, 0, 500 scene coordinates.

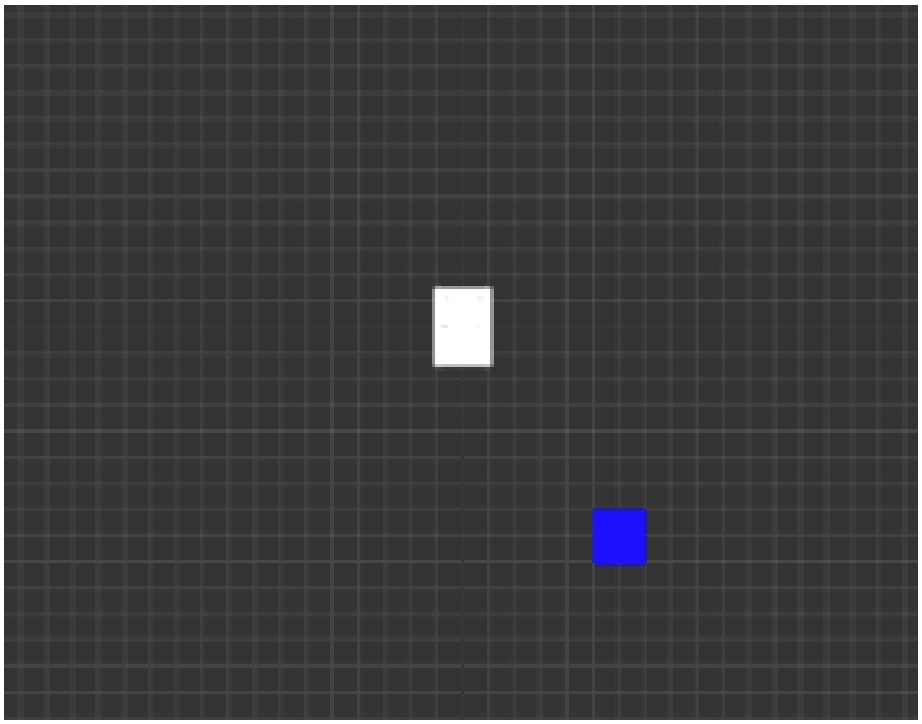

FIGURE 13. Blue Lidar point in the bottom right corner with Dolly™ in the centre.

The user can return into the main menu from the map view and pick the manual control option. This view, located at 0, 0, -500 scene coordinates, brings the user to the manual control view (FIGURE 14). This view allows the user to see a rotating model of the Dolly™ mobile robot while giving the option for the user to see and interact with menus and options. Manually sending movement commands to the robot can also be achieved here by using the keyboard arrow keys.

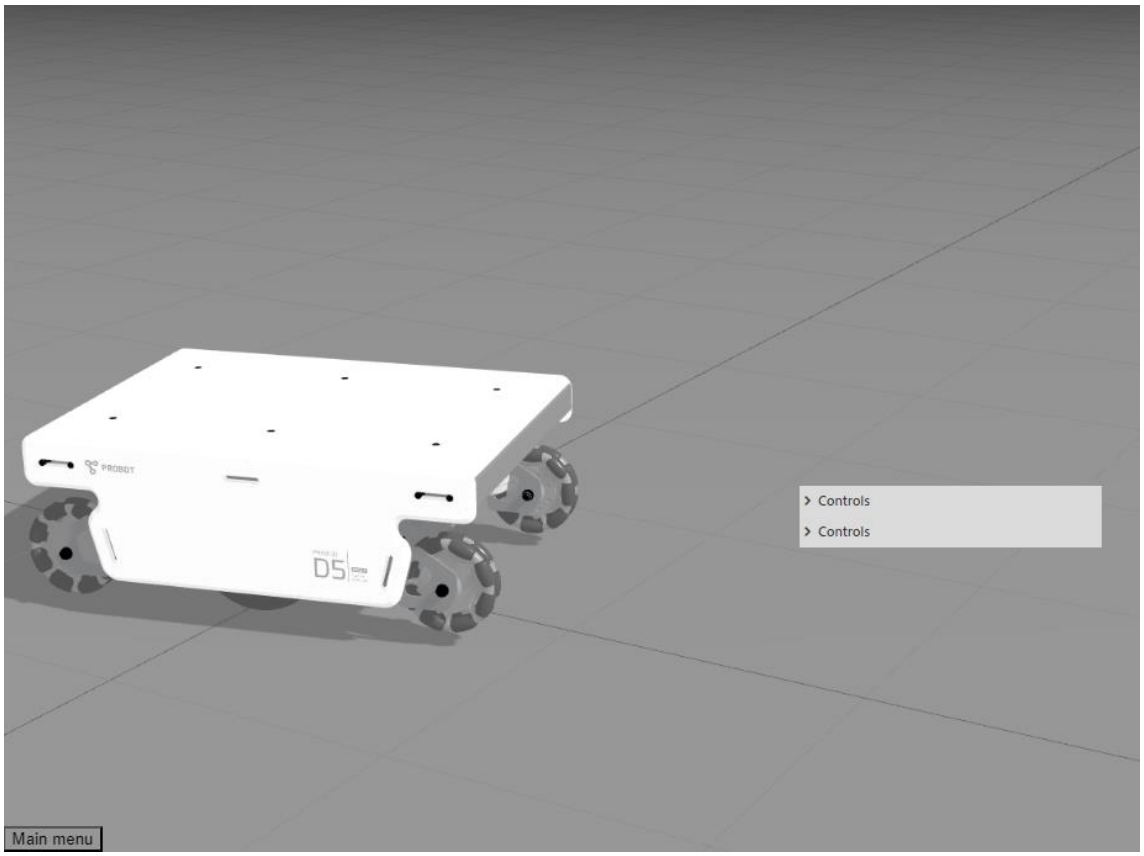

FIGURE 14. Manual control view in the scene.

4 CONCLUSION

The main objective for this thesis was to create a POC of a mobile robot user interface using Three.js and JavaScript and analyse the creation of 3D graphics within the web environment. The suitability of Three.js was analysed in the process of creating a 3D user interface. This thesis introduced some of Three.js's main features along with practical examples in the form of an UI.

The UI consists of a 3D scene where all the objects would reside in. Main menu, manual control and loaded map views were implemented. The main menu provided a portal to access manual control mode, which included a 3D model of Dolly™ along with a couple of menus. This is the view that is intended to be used for manually controlling the robot with arrow keys or visual joysticks. The second main map view was intended to be used as a view that loads or creates a map of the robot's surroundings. Waypoints, movement paths and Lidar scan points can be added, and the view can be inspected from different angles with the use of Three.js's camera controls.

Without previous experience with Three.js, the quick testing and creation of components was made possible by vast documents and great support from the community and development environments. The Three.js website provided easy access to explanations and practical and clean code examples. The ability to load external 3D models simplified the creation of the first examples within the development which included a spinning model of Dolly™, which was later further modified and placed in the manual control view for visualization purposes.

The project was viewed as a success and as the company's first real test dive into the world of 3D web graphics with Three.js. With the focus on Three.js and its functionality and ease of development, it was concluded that Three.js is a suitable technology choice for creation of a mobile robot UI. The components developed and the knowledge learned will be implemented to future UI projects within the company. This thesis and its results provided valuable insight into UI development, its requirements, and the technologies allowing the company to focus more resources in Three.js development.

The project offers opportunities for future expansion. An extensive list of potential features was initially created, from which I selected the most crucial ones to develop a POC. Many of the ideas presented on the original list were later found to be possible with the use of Three.js during research

and development. Several VR libraries implemented on Three.js were found and for future projects, VR UIs could be researched and implemented. At this point, the user interface is visually very rough, but it will be drastically improved in the upcoming versions when UI development will be fully initiated.

My focus with this thesis was directed to designing, testing, and implementing the envisioned features as components after which they would be implemented one after the another into the production version with the lead of our CTO. These features mentioned in the thesis were the base for starting the production version development with minor changes to the functionalities of them. The features formed the initial POC UI within Three.js and brought into attention what is possible and feasible to create with the technologies involved in this project. The UI is simple to use, however requires much more computing power to run properly compared to more traditional 2D user interfaces. Even at this stage, the 3D aspect of the UI brings the possibility to analyse the situation and location of the robot from different angles providing a clearer and more accurate representation of the robots' surroundings. This improves the usability and provides additional value to the mobile robot.

REFERENCES

Aucoin, Nicole & Jenkin, Michael 2000. An Immersive 3D User Interface for Mobile Robot Control. Search date 13.11.2023. https://www.researchgate.net/publication/2947207_An_Immersive_3D_User_Interface_for_Mobile_Robot_Control

Badler, Norman I. "3D Object Modeling Lecture Series" (PDF). University of North Carolina at Chapel Hill. Archived (PDF) from the original on 2013-03-19. Search date 19.12.2023 http://gamma.cs.unc.edu/courses/graphics-s09/LECTURES/3DModels_SurveyPaper.pdf

Favata, Dom 6.10.2023. EXPLORING THE WORLD OF 3D GRAPHICS WITH WEBGL: A COMPREHENSIVE GUIDE. Jahnel Group, Inc. Search date 14.11.2023. https://insights.jahnelgroup.com/exploring-the-world-of-3d-graphics-with-webgl-a-comprehensive-guide

Gordon, Jawara 1.5.2023. JavaScript 3D Modeling with Three.js. medium.com. Search date 14.11.2023. https://medium.com/@jawaragordon/javascript-3d-modeling-with-three-js-a169aa2694e6

Interaction Design Foundation 2016. What is User Interface (UI) Design? Search date 6.11.2023. https://www.interaction-design.org/literature/topics/ui-design

Jet Propulsion Laboratory, California Institute of Technology. User Interfaces. Search date 13.12.2023. https://www-robotics.jpl.nasa.gov/what-we-do/applications/user-interfaces/

Khronos Group Inc 2023. WebGL - Low-Level 3D Graphics API Based on OpenGL ES. Search date 7.11.2023. https://www.khronos.org/webgl/

MAP Systems. A Detailed Guide to 3D Textures. Search date 22.12.2023. https://mapsystemsindia.com/resources/what-is-3d-texturing.html

Omron 2023. LD60/90. Search date 18.10.2023. https://industrial.omron.fi/fi/products/ld-series

Podmurnyi, Slava 6.10.2021. 6 Best WebGL Libraries for Perfect 3D Web Graphics. hackermoon.com. Search date 14.11.2023. https://hackernoon.com/6-best-webgl-libraries-for-perfect-3d-web-graphics

Suomen Robotiikkayhdistys ry 2023. Teollisuuden robotiikka. Helsinki: Keuruun Laatupaino KLP Oy 2023.

Thomas Sifferlen 2021. Chapter 1 – The Basic Concepts of Three.js. Three.js University. Search date 7.11.2023. https://en.threejs-university.com/2021/08/02/chapter-1-the-basic-concepts-of-three-js/

Threejs.org 2024. "OrbitControls". Search date 12.1.2024. https://threejs.org/docs/#examples/en/controls/OrbitControls

Threejs.org 2023. "Loading 3D Models". Search date 14.11.2023. https://threejs.org/docs/#manual/en/introduction/Loading-3D-models

Threejs.org 2024. Fundamentals. Search date 16.1.2024. https://threejs.org/manual/#en/fundamentals

Wikipedia 2023. 3D computer graphics. Search date 6.11.2023. https://en.wikipedia.org/wiki/3D_computer_graphics

Wikipedia 2023. 3D modeling. Search date 21.12.2023. https://en.wikipedia.org/wiki/3D_modeling

Wikipedia 2023. 3D rendering. Search date 19.12.2023 https://en.wikipedia.org/wiki/3D_rendering

Wikipedia 2023. UV mapping. Search date 29.12.2023. https://en.wikipedia.org/wiki/UV_mapping

Wikipedia 2023. User interface. Search date 6.11.2023. https://en.wikipedia.org/wiki/User_interface

Wikipedia 2023. Texture mapping. Search date 29.12.2023. https://en.wikipedia.org/wiki/Texture_mapping

Wikipedia 2023. Computer-aided design. Search date 21.12.2023. https://en.wikipedia.org/wiki/Computer-aided_design

Wikipedia 2023. Graphical user interface. Search date 6.11.2023. https://en.wikipedia.org/wiki/Graphical_user_interface