COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

Design and development of a guidance system

Ana Gortázar Florit

Director: Dr. Antonio García y Garmendia

Madrid

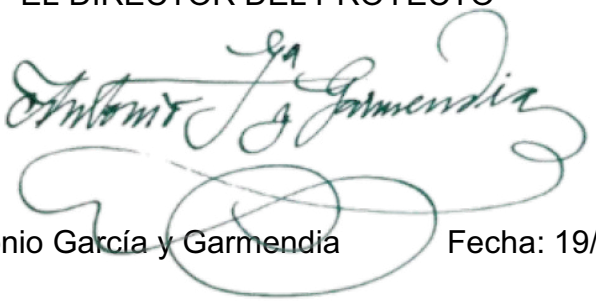
Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título Diseño y desarrollo de un sistema de guía

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2023/2024 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Ana Gortázar Florit Fecha: 19/07/2024

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO



Fdo.: Dr. Antonio García y Garmendia Fecha: 19/07/2024

BACHELOR'S DEGREE IN INDUSTRIAL TECHNOLOGY ENGINEERING

FINAL THESIS PROJECT

Design and development of a guidance system

Ana Gortázar Florit

Director: Dr. Antonio García y Garmendia

Madrid

DISEÑO Y DESAROLLO DE UN SISTEMA GUÍA

Autor: Ana Gortázar Florit.
Director: Dr. Antonio García y Garmendia.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

El proyecto consiste en el diseño y desarrollo de un dispositivo de asistencia para ayudar a las personas con discapacidad visual a navegar de manera segura en entornos urbanos. El primer paso fue analizar el estado del arte actual, enfocándose especialmente en el estudio de patentes relacionadas con el tema. El objetivo de este paso inicial era ver las áreas de mejora dentro de la industria y entender mejor el funcionamiento de estos dispositivos para que el producto diseñado fuera el mejor posible. Posteriormente, se desarrolló la teoría detrás del funcionamiento del dispositivo, con un enfoque especial en los componentes tecnológicos, cómo se tendrían que colocar y utilizar para lograr un sistema de guía seguro y las ecuaciones necesarias para que todo funcionase. Después la teoría se llevó a la práctica mediante la creación de una simulación 3D en el software llamado Webots, la cuál ha sido controlada mediante un código en el lenguaje de Python. Aquí se comprobó que los teoremas teóricos funcionaban de manera correcta. Después, se analizaron los aspectos económicos relacionados con la creación y comercialización de este dispositivo. Aquí se encontró que el sistema desarrollado no era tan solo económicamente viable, sino que los beneficios de su venta podrán ser enormes. Finalmente, se comentaron las conclusiones y los resultados, verificando que los objetivos propuestos en un principio eran cumplidos.

Palabras clave: ETA, detección de obstáculos, navegación, simulación.

1. Introducción

La continua evolución de la tecnología ha impactado profundamente todos los aspectos de la vida humana, haciéndola más accesible e inclusiva para personas con discapacidades. Entre estos avances, los dispositivos de asistencia para ciegos han visto un desarrollo significativo. Con la sociedad envejecida de hoy, el número de personas con dificultades visuales aumenta cada año y, por lo tanto, la necesidad de seguir innovando crece. Este proyecto se centra en diseñar una nueva solución que ayude en la navegación segura de entornos urbanos.

2. Definición del Proyecto

El objetivo de este proyecto es diseñar un sistema de guía para ayudar a las personas con discapacidad visual a navegar en entornos urbanos al aire libre. Este sistema debe ser capaz de detectar obstáculos y evitarlos, encontrar el camino óptimo al destino deseado por el usuario y guiarlos, identificar escaleras y semáforos y proporcionar la retroalimentación necesaria al usuario. Para lograr estas características y hacerlo de la

manera más efectiva, reduciendo el costo y la complejidad, el proyecto ha seguido una metodología estructurada que consiste en análisis del estado del arte, diseño del dispositivo, simulación, viabilidad económica y resultados.

En el primer paso del estado del arte, fue esencial identificar los diferentes tipos de ayudas de navegación asistida, así como las tecnologías exactas que utilizaban. Para esto, la fuente principal de información fueron las patentes relacionadas con el tema en las que se podía leer y entender todo el aspecto técnico.

Después de este primer paso de familiarización con la tecnología y otras ayudas electrónicas de viaje o ETAs por sus siglas en inglés, el sistema se desarrolló teóricamente. En esta fase del proyecto, el objetivo era crear el dispositivo más simplista posible, principalmente para reducir costos, prolongar la vida de la batería y reducir las posibilidades de errores. Para esto, se confió en las patentes mencionadas anteriormente, pero también hubo un proceso de invención. La necesidad de reducir la complejidad se convirtió en la necesidad de crear nuevas soluciones, por ejemplo, en la posición de los componentes tecnológicos, la detección de escaleras o en el proceso de evitación de obstáculos.

Con la teoría desarrollada y el principal obstáculo de complejidad y costo superado, se realizó una simulación para confirmar el correcto funcionamiento del sistema. El principal desafío en este paso fue familiarizarse con el software Webots y Python, dos mecanismos de simulación y codificación que nunca había utilizado antes. Una vez superada la curva de aprendizaje, se creó la simulación City World, utilizando los objetos ya ofrecidos en Webots y diseñando un modelo 3D del ETA desde cero. Con esta simulación, se probaron todos los escenarios posibles y se confirmó que la teoría desarrollada podría implementarse en la realidad.

En la parte final de este proyecto se realizó un estudio de viabilidad económica, donde se encontró que la ayuda de navegación creada sería extremadamente competitiva y, por lo tanto, rentable en el mercado actual.

3. Descripción del Sistema

El dispositivo creado está formado por siete módulos diferentes que trabajan juntos para lograr los objetivos establecidos para el proyecto. El siguiente diagrama de flujo muestra cómo estos módulos interactúan entre sí y las señales necesarias para esa comunicación.

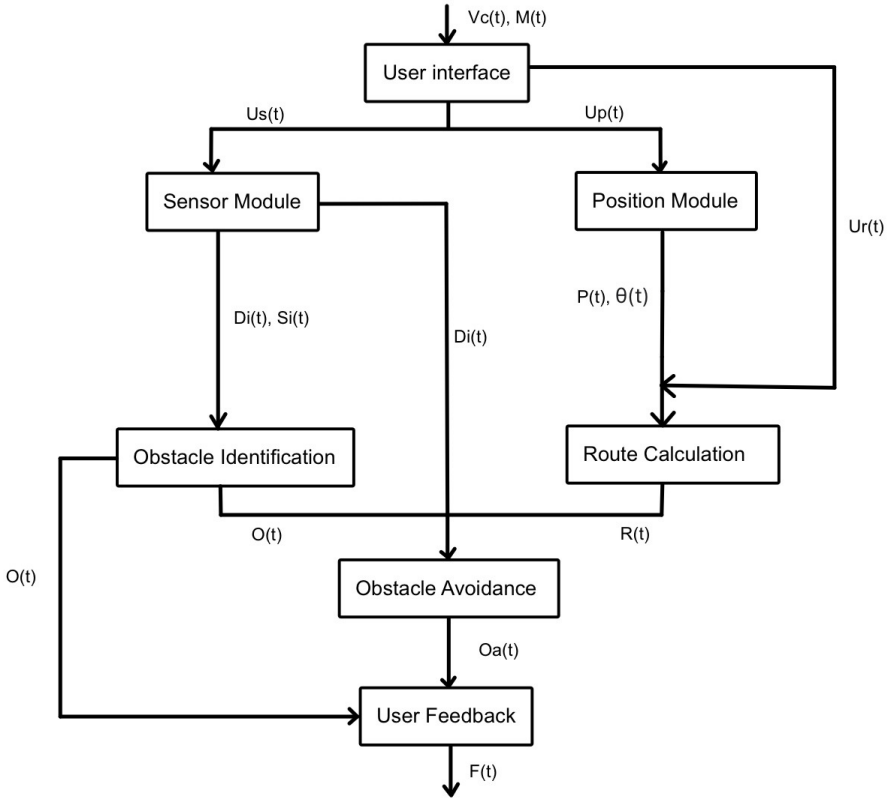


Figura 1: Diagrama de flujo del sistema. Fuente: Elaboración propia (2024).

El módulo de interfaz de usuario permite al usuario encender el dispositivo, así como ingresar su destino deseado. Siguiendo el lado izquierdo del diagrama de flujo, encontramos que la señal de encendido va al módulo de sensores, que está compuesto por cuatro sensores ultrasónicos, un LiDAR 2D y una cámara RGB. Tres de los sensores ultrasónicos están diseñados para detectar obstáculos que están por debajo de la cintura del usuario, uno de ellos está posicionado para detectar obstáculos aéreos, el LiDAR 2D se utiliza para detectar objetos a nivel de la cintura y la cámara se usa para identificar semáforos.

La información del sensor luego va al módulo de identificación de obstáculos que está diseñado para identificar dos tipos de obstáculos: escaleras y semáforos. Para la detección de escaleras hacia arriba se utiliza un sensor ultrasónico, y para las escaleras hacia abajo se utilizan los sensores ultrasónicos dos y tres. La lógica de detección es muy simple y consiste en comparar las distancias devueltas con las esperadas para el alza y la pisada de un escalón en una escalera; si coinciden, se puede confirmar la identificación

[1]. Para la detección de semáforos se utiliza la escala de colores HSV, que diferencia entre tono, saturación y valor haciendo que la luz LED en las señales de tráfico destaque [2]. Al determinar los rangos correctos para rojo y verde, se pueden determinar los colores y, por lo tanto, el estado del semáforo. Para asegurar una lógica robusta, también se utilizan los contornos y formas de esos colores, verificando si forman un círculo o una figura humana.

Siguiendo el lado derecho del diagrama de flujo, encontramos el módulo de posición; este módulo consiste en un GPS y una Unidad de Medición Inercial (IMU). El GPS es capaz de conocer la posición del usuario en todo momento y la IMU identifica la orientación del usuario al medir su ángulo de guiñada. Estas señales, así como el destino deseado ingresado, se transmiten al módulo de cálculo de ruta. Este módulo utiliza el algoritmo A* para determinar el camino más corto y, por lo tanto, óptimo a esa ubicación deseada[3]. Para que este algoritmo funcione, el sistema necesita tener acceso a un mapa de nodos y bordes, un nodo es una posición en el mapa y un borde es un camino transitable. Cuando se ingresa una ubicación, el algoritmo analiza todos los posibles caminos y calcula el costo de ellos utilizando la distancia euclidiana; por lo tanto, el camino con el costo mínimo es de hecho el camino óptimo.

Ambos lados del diagrama de flujo convergen en el módulo de evitación de obstáculos o también entendido como módulo de movimiento, formado por un motor y un controlador de motor. Este módulo recibe toda la información, la analiza y luego envía el movimiento al controlador del motor. Aquí están los diferentes escenarios y comandos posibles:

- Semáforo Rojo Detectado: Cuando se detecta un semáforo rojo, hay dos posibles escenarios. El primero es una detección simple que causará que se envíe un comando de movimiento de parada hasta que se detecte un semáforo verde. El segundo ocurre cuando se detecta un semáforo rojo en medio del paso de peatones, lo que significa que el usuario comenzó a cruzar cuando era seguro hacerlo y luego el semáforo cambia. En este caso, el comando de movimiento de parada se sobrescribe y el dispositivo continuará avanzando.

- Nodo del Algoritmo A*: Para que el dispositivo siga el camino correcto en general, se implementa un método de seguimiento a menor escala. El sistema va de un nodo a la vez y calcula y recalcula constantemente el ángulo de giro necesario para alcanzar ese próximo objetivo.

- Nodo de Obstáculo: La lógica de evitación de obstáculos es crear dos nodos temporales, uno junto al obstáculo para que el ETA gire y otro después del obstáculo para que vuelva al camino. Estos nodos se establecen en función de la ubicación del obstáculo en relación con la posición del usuario, lo que significa que, si el objeto

[1] Bouhamed et al (2013)
[2] Wonghabut, Pasit, et al (2018).
[3] Oxford (retrieved in 2024).

está más a la izquierda, el nodo se crea a la derecha y viceversa. Para que esto funcione, el comando de movimiento se sobrescribe, por lo que en lugar de ir al siguiente waypoint en el algoritmo A*, va al siguiente waypoint en el algoritmo de evitación de obstáculos.

El módulo final es la retroalimentación del usuario, que se utiliza para transmitir toda la información necesaria al usuario a través de un comando de voz, pero sin mandar información excesiva.

Para que estos módulos funcionen como se teoriza, es esencial desarrollar correctamente el diseño físico y colocar cuidadosamente los componentes tecnológicos. El diseño físico final y la colocación de componentes se muestra en la siguiente imagen.

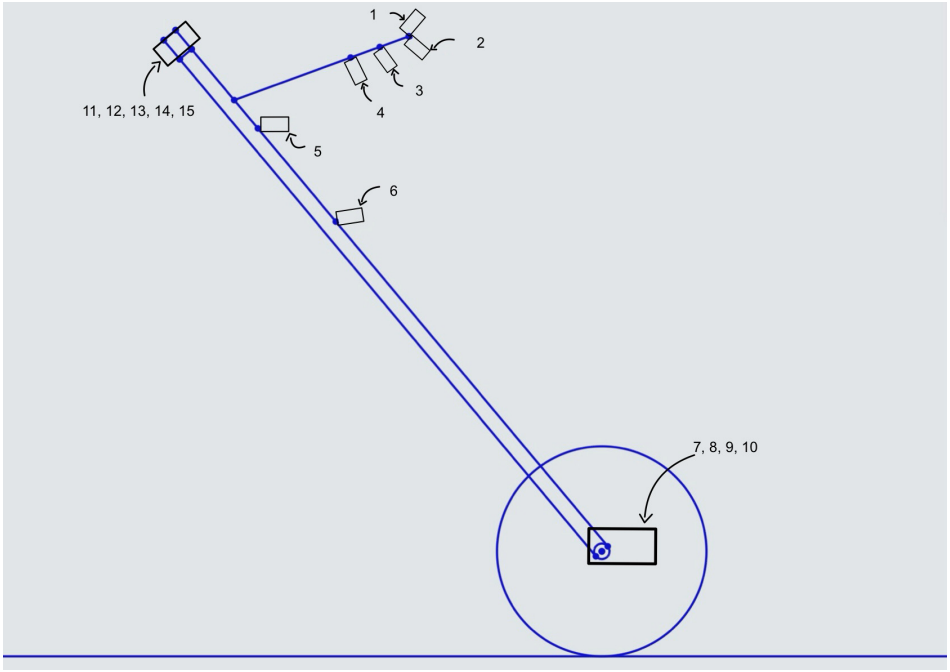


Figura 2: Posicionamiento de componentes. Fuente: Elaboración propia (2024).

4. Conclusiones

En conclusión, se ha creado un dispositivo de asistencia innovador para ciegos, capaz de hacer lo siguiente:

- Detectar escaleras ascendentes y descendentes.
- Conocer la ubicación de los cruces peatonales e identificar el color del semáforo, para determinar si es seguro cruzar.
- Determinar el camino óptimo al destino deseado por el usuario y guiarlos hasta él mientras se mantienen en la acera.
- Detectar obstáculos desde el nivel del suelo hasta el nivel de la cabeza.
- Evitar obstáculos cuando se detectan en el camino del usuario.

Con la simulación en Webots y la creación de diferentes escenarios, se pudieron probar situaciones de la vida real que confirmaron el correcto funcionamiento del dispositivo. Además, de funcionar correctamente, el estudio de viabilidad económica concluyó que el proyecto es factible y potencialmente muy rentable

Bibliografía

[1] Bouhamed, S. A., Kallel, I. K., & Masmoudi, D. S. (2013). New electronic white cane for stair
case detection and recognition using ultrasonic sensor. *International Journal of Advanced Computer Science and Applications, 4(6)*

[2] Wonghabut, P., et al. (2018). Traffic light color identification for automatic traffic light violation
detection system. In *2018 International Conference on Engineering, Applied Sciences, and Technology (ICEAST)* (pp. 1-4). IEEE.

[3] Math.oxford.emory.edu (retrieved in 2024). *A* Algorithm*. Retrieved from: https://math.oxford.emory.edu/site/cs171/theAStarAlgorithm/. Accessed: July, 2024.

DESIGN AND DEVELOPMENT OF A GUIDANCE SYSTEM

Author: Ana Gortázar Florit.
Supervisor: Dr. Antonio García y Garmendia.
Collaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

The project consists of designing and developing an assistance device to help visually impaired people navigate safely in urban environments. The first step was to analyze the current state of the art, with a particular focus on studying patents related to the topic. The objective of this initial step was to identify areas for improvement within the industry and to better understand the functioning of these devices to ensure the designed product would be the best possible. Subsequently, the theory behind the device's operation was developed, with a special focus on the technological components, how they should be placed and used to achieve a safe guidance system, and the necessary equations for everything to work. The theory was then put into practice by creating a 3D simulation in software called Webots, which was controlled using Python code. This demonstrated that the theoretical theorems worked correctly and could be applied in real life. Next, the economic aspects related to the creation and commercialization of this device were analyzed. It was found that the system developed was not only economically viable, but the potential sales benefits could be enormous. Finally, the conclusions and results were discussed, verifying that the initially proposed objectives were met.

Keywords: Electronic travel aid (ETA), obstacle detection, navigation, simulation

1. Introduction

The continuous evolution of technology has profoundly impacted all aspects of human life, making it more accessible and inclusive for individuals with disabilities. Among these advancements assistive devices for the blind have seen a significant development. With today's aging society the number of people with visual difficulties increases each year and therefore the need to keep innovating grows. This project focuses on designing a new solution that aids in the safe navigation of urban environments.

2. Project definition

The objective of this project is to design a guidance system to help visually impaired individuals navigate outdoor urban environments. This system must be able to detect obstacles and avoid them, find the optimal path to the user's desired destination and guide them, identify stairs and traffic lights and give the necessary feedback to the user. To achieve these characteristics and do so in the most effective way, reducing the cost

and complexity the project has followed a structured methodology consisting of state of the art analysis, device design, simulation, economic viability and results.

In the first step of state of the art, it was essential to identify the different types of navigational assistive aids, as well as the exact technologies they used. For this, the main source of information were patents related to the topic in which all of the technical side of things could be read and understood.

After this first step of familiarizing oneself with the technology and other Electronic Travel aids or ETAs for short, the system was then developed theoretically. In this phase of the project, the aim was to create the most simplistic device possible, mainly in order to reduce costs, prolong battery life, and reduce the possibilities of errors. For this, there was a reliance on the previously mentioned patents but there was also a process of invention. The need to reduce complexity turned into the need to create new solutions, for example in the positioning of technological components, the detection of staircases or in the obstacle avoidance process.

With the theory developed and the main hurdle of complexity and cost overcome, a simulation was done to confirm the system's correct functionality. The main challenge with this step was getting familiarized with the Webots software and Python, two simulation and coding mechanisms that I had never previously used. Once the learning curve was overcome the actual simulation City World was created, using the objects already offered in Webots and designing a 3D model of the ETA from scratch. With this simulation all of the possible scenarios were tested, and it was confirmed that the theory developed could be implemented in reality.

In the final part of this project an economic viability study was done, where it was found that the navigational aid created would be extremely competitive and therefore profitable in today's market.

3. Device Description

The device created is formed by seven different modules that work together to achieve the objectives set for the project. The following flow chart shows how these modules interact with one another and the signals that are needed for that communication.

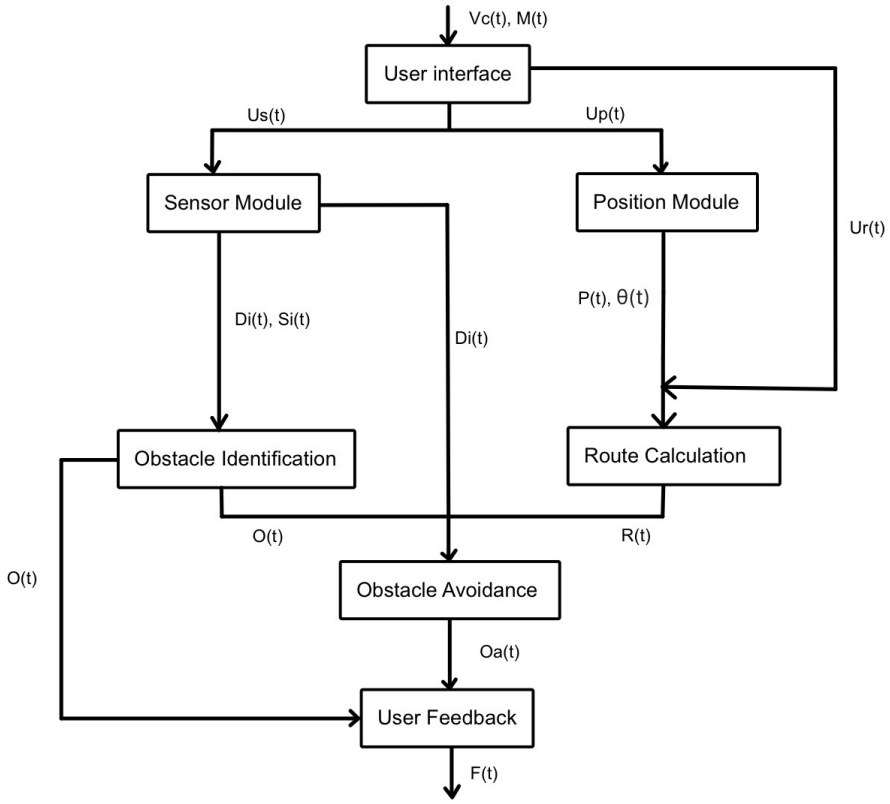


Figure 3: System flow chart. Source: Own elaboration (2024).

The User Interface Module allows the user to turn on the device, as well as input their desired destination. Following the left side of the flow chart we find that the turn on signal goes to the sensor module, which is made up of four ultrasonic sensors, one 2D LiDAR and an RGB camera. Three of the ultrasonic sensors are designed to detect obstacles that are below the user's waist, one of them is positioned to detect overhead obstacles, the 2D LiDAR is used to detect waist level objects and the camera is used to identify traffic lights.

The sensor information then goes to the obstacle identification module which is designed to identify two types of obstacles stairs and traffic lights. For the upwards stairs detection ultrasonic sensor, one is used and for downwards stairs ultrasonic sensors two and three are used. The detection logic is incredibly simple and consists in comparing the distances returned with those expected for the rise and tread of a step in a staircase, if they match the

identification can be confirmed [4]. For the traffic light detection, the HSV color scale is used, which differentiates between hue, saturation and value making the LED light in traffic signs stand out. By determining the correct ranges for red and green the colors and therefore traffic light state can be determined [5]. To ensure a robust logic, the contours and shapes of those color are also used, by checking if they form a circle or a human.

Following the right side of the flow chart we find the position module; this module consists of a GPS and an Inertial Measurement Unit (IMU). The GPS is able to know the user's position at all times and the IMU identifies the user's orientation by measuring its yaw angle. These signals, as well as the inputted desired destination are then transmitted to the route calculation module. This module uses A* algorithm to determine the shortest and therefore optimal path to that desired location [6]. For this algorithm to work the system needs to have access to a waypoint and edge map, a waypoint is a position in the map and an edge is a walkable path. When a location is inputted the algorithm analysis all of the possible paths and calculates the cost of them by using the Euclidean distance, therefore path with the minimum cost is in fact the optimal path.

Both sides of the flow chart converge in the obstacle avoidance module or also understood as movement module is formed by a motor and motor driver. This module receives all of the information, analyzes it and then sends the movement to the motor driver. Here are the different possible scenarios and commands:

- Red Traffic Light Detected: When a red traffic light is detected there are two possible scenarios. The first is a simple detection which will cause a stop movement command to be send until a green light is detected. The second happens when a red light is detected in the middle of the crosswalk, meaning the user starting crossing when it was safe to do so and then the light changes. In this case the stop movement command is overwritten, and the device will continue to move forward.

- A* Algorithm Waypoint: In order for the device to follow the overall correct path a lower scale following method is implemented. The system goes one waypoint at a time and constantly calculates and recalculated the necessary turn angle to reach that next goal.

- Obstacle Waypoint: The obstacle avoidance logic is to create two temporary waypoints, one next to the obstacle so the ETA will turn and another after the obstacle so it will go back to the path. These waypoints are set based on the location of the obstacle in relation to the user's position, meaning if the object is more to the left the waypoint is created to the right and vice versa. For this to work the movement command is overwritten, so instead of going to the next waypoint in the A* algorithm it goes to next waypoint in the obstacle avoidance algorithm.

[4] Bouhamed et al (2013)
[5] Wonghabut, Pasit, et al (2018).
[6] Oxford (retrieved in 2024).

The final module is the user feedback, this is used to transmit all of the necessary information to the user via a voice command, but without overdoing it.

In order for these modules to work as theorized it is essential to correctly develop the physical design and to carefully place the technological components. The final physical design and component positioning is shown in the following image.

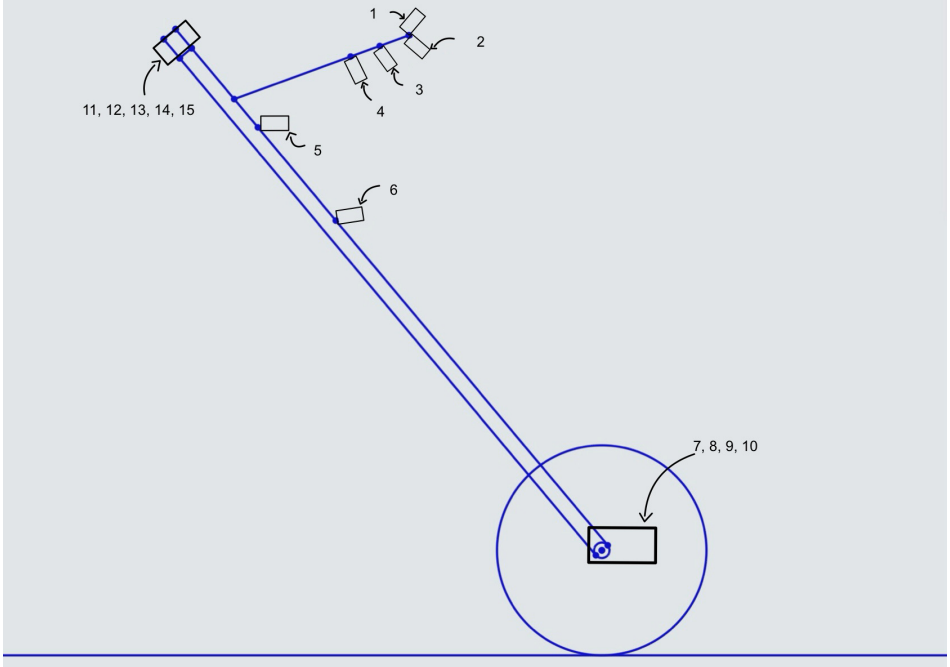


Figure 4: Positioning of components. Source: Own elaboration (2024).

4. Conclusions

In conclusion an innovative assistive device for the blind has been created, capable of doing the following:

- Detecting upward and downward staircases,
- Knowing the location of crosswalks and identifying the traffic light color, to determine whether or not it is safe to cross.
- Determining the optimal path to the user's desired location and guiding them to it while staying on the sidewalk.
- Detecting obstacles from ground level, all the way up to head level.
- Avoiding obstacles when they are detected to be in the user's path.

With the Webots simulation and the creation of different scenarios, real life situations were able to be tested which confirmed the correct functioning of the device. In adittion, to

working correctly the economic viability study concluded that the project is in fact feasible and potentially very profitable.

Bibliography

[1] Bouhamed, S. A., Kallel, I. K., & Masmoudi, D. S. (2013). New electronic white cane for stair
case detection and recognition using ultrasonic sensor. *International Journal of Advanced Computer Science and Applications, 4(6)*

[2] Wonghabut, P., et al. (2018). Traffic light color identification for automatic traffic light violation
detection system. In *2018 International Conference on Engineering, Applied Sciences, and Technology (ICEAST)* (pp. 1-4). IEEE.

[3] Math.oxford.emory.edu (retrieved in 2024). *A* Algorithm*. Retrieved from: https://math.oxford.emory.edu/site/cs171/theAStarAlgorithm/. Accessed: July, 2024.

Acknowledgments

I would like to thank my tutor, Dr. Antonio García y Garmendia, for helping me through this process to elaborate the best possible thesis. I would also like to thank my parents for their support during these past few months.

Index

Figure Index

Table Index

Chapter 1. INTRODUCTION

1.1 MOTIVATION

The main motivation for choosing this project comes from a desire to make a meaningful impact on the world and help others in need, specifically those with a visual impairment. As stated in the introduction, the number of people affected by this disability grows every year and thus it is imperative that the technology designed to assist them evolves. Addressing this issue is not just about accessibility but about equality and giving everybody the same opportunities to create a more inclusive and compassionate society.

Even though there have been significant advancements in this field, assistive devices for the blind still have important limitations, for example the fact that they are non-adaptable. As time passes the challenges that the visually impaired face grow and change, but there is no technology able to adapt. Therefore, these assistive devices must constantly evolve an improve in order to keep up with the needs of the blind. The two main constraints are cost-effectiveness and user-friendliness [1]. ETAs are extremely costly which is a very worrying problem since 60% of blind Americans are unemployed and this technology can have price between $1,000 and $6,000 [2]. Consequently, even if ETAs were able to meet all of the needs required to move independently most of the population would not be able to afford them. For this very reason the motivation behind this paper grows given the importance of making a device that is accessible to everyone.

In addition, as an Industrial Engineering student with a focus on electronics I am committed to further improving my programming skills. This project is the perfect opportunity to really expand my knowledge and apply it to a real project that could help thousands.

[1] Masal et al (2024).
[2] American Community Survey (retrieved in 2024).

1.2 OBJECTIVES

The main objective of this project is to develop an ETA that guides visually impaired individuals so they can navigate from point A to point B as independently as possible, similarly to a nondisabled person. To achieve this, the system must meet the following specific requirements.

1.2.1 OBSTACLE DETECTION

Obstacle detection is an essential requirement for any Electronic Travel Aid, since it ensures the safety and mobility of the user. Within obstacle detection, there are different functions that must be analyzed:

- **Range of Detection:** The range has two axes that need to be considered: the y-axis and the x-axis. For the y axis, the goal is to create a device that can detect objects from the ground all the way to the user's head. There is a special focus on high obstacles, such as tree branches or signs, where a traditional white cane would fall short. For the x-axis, the aim is to achieve the widest range possible so that all the surrounding objects can be detected, ideally covering 360 degrees.

- **Static and Dynamic Obstacles:** While it is essential to detect static objects, dynamic obstacles must not be overlooked. This project there will place special emphasis on moving objects since this ETA is especially designed to be used outdoors. The device must be able to identify and avoid static and moving obstacles alike.
- **Obstacle Identification:** The system should be capable of identifying two types of obstacles, stairs and traffic lights. This will enhance user safety and allow the device to guide the individual correctly.

- **Distance Measurement:** The device must accurately calculate the distance between the user and the detected objects. This is crucial in order to avoid collisions and ensure safe navigation.

- **Range of Distance Detection:** Within the context of distance calculation, the objective is to make the range at which the system can accurately measure distances as wide as possible. Ideally, it should be able to measure objects from a few centimeters to several meters away. The greater this range, the safer the ETA will be for the user.

- **Depth Perception:** It is not enough for the device to calculate how far away objects are; it must also determine the depth of them. This capability will allow the system to distinguish between different objects and guide the user through a safe path.

- **Time processing:** All of these objectives are meaningless if they are not achieved within a short timeframe. Objects must be detected in real time, so the delay must be minimized as much as possible. The shorter the delay, the more accurate and safer the ETA will be.

1.2.2 GEOLOCATION AND NAVIGATION

To guide the user safely from point A to point B, the device must be able to determine the user's exact location (longitude and latitude) and devise an optimal route to get there. This requires the ETA to have the following functions:

- **Position:** The device must be able to determine the user's exact location in real time. Without this function, the ETA would be completely ineffective when guiding the user outdoors, which is the main focus of this project. Real time data allows the system to update the individual's location and ensure accurate navigation.

- **Self-orientation:** The device must know the direction the user is facing, which is essential for the user to correctly interpret the directions given.

- **Route Planning:** Similarly to Google Maps, this device must calculate the best path from the user's location to their chosen destination. It involves analyzing the possible routes and choosing the one that minimizes distance and avoids obstacles, therefore optimizing travel time.

1.2.3 USER FEEDBACK

The data gathered by the ETA, from both obstacle detection and location, must be effectively transmitted to the user to ensure their safety. This feedback, along with the actual guiding, is what allows the visually impaired individual navigate their outdoor surroundings effectively. Here are some aspects that must be taken into account:

- **Detailed Feedback:** The information provided to the user must be detailed enough to help them take the best course of action to ensure their safety. For example, to make appropriate decisions they must know whether the obstacle is

a traffic sign, a pedestrian walking by or a car at full speed. However, it is also very important to avoid providing excessive feedback that might confuse the user, there must be a balance.

- **Intuitive:** The device as a whole but especially the feedback must be intuitive, minimizing the learning curve as much as possible. The easier the ETA is to use and understand, the more accessible it will be to people who might find it challenging to learn new technology systems, such as cognitive impaired individuals or the elderly. This will also improve the safety, as it will reduce user errors.

- **Real Time Feedback:** The system must provide real time information about the user's surroundings and potential risks, minimizing any delays. The faster this information is given, the safer and more efficient the navigation will be, as it ensures that the individual has enough time to act.

1.2.4 STAIR NAVIGATION

An essential requirement for this project is that the ETA must be able to ascend and descend stairs on its own. This is crucial to give the blind individual more freedom and independence. To achieve this, the following must be taken into account:

- **Stair Detection:** The ETA must accurately detect the presence of a staircase and determine the height, depth, and edge of each step.

1.2.5 PORTABILITY

Lastly, the ETA must be lightweight and easy to carry by a single person to ensure it can be transported easily using different types of transportation, including cars, buses and planes. Here are the key aspects that need to be considered:

- **Lightweight:** The device must be light enough to be lifted and carried by individuals of all ages and physical abilities. To achieve this, the materials selected must have a low density but still be durable. Additionally, the sensors must be as lightweight as possible.

- **Compact Size:** The ETA must also be compact enough to fit into different transportation vehicles, making it even more convenient for the user. It should be designed to occupy as little space as possible and include foldable features

wherever possible to reduce its size even further when not in use. Therefore, the sensors and other components should be small and efficiently arranged.

All of the requirements stated above align with the rules of ETAs according to the National Research Council [3], which determine whether or not a specific technology is considered an electronic travel aid or not. These rules are:

1. Determining obstacles around the user's body from the ground to the head.

2. Affording some instructions to the user about the surface, for example gaps or textures.

3. Finding items surrounding the obstacles.

4. Providing information about the distance between the user and the obstacle with essential direction instructions.

5. Proposing notable sight locations in addition to identification instructions.

6. Affording information to give the ability of self-orientation and mental map of the surroundings.

Finally, this product will aim to achieve two out of the seventeen goals stated in the United Nations Sustainable Development Goals for 2030. These are good health and well-being and reduced inequalities [4].

In conclusion, the end goal is for the device to guide the individual to allow them to go from point A to point B as a nondisabled person would.

1.3 METHODOLOGY

To begin with the project the first step must be to determine the exact sensors and technologies that must be implemented into the device for it to achieve the objectives set in the previous section. In order to do so a more detailed investigation will be done to select the best components for our goals. After doing so there will be a very detailed explanation on how each of them works, how they help reach the objectives set and how they will be implemented in the device.

Then the physical design of the prototype will be made, taking into account all the sensors it must have and where they should be located to function correctly. Furthermore,

[3] Blasch et al (1997).
[4] United Nations (retrieved in 2024).

in this section there will be an important focus on the properties that the wheel/s must have. This will be an extensive process since they must be able to go up and downstairs, which is a very difficult problem that has to be addressed properly. Once finished with the design, the result should be a new and innovative device that satisfies all of the objectives. For this to be successful both the mechanical and electronical properties have to be considered.

The next step will be to actually program all the components of the ETA, and this will be done using Python. After a simulation will be created to show how the device works and most importantly that it does indeed work.

The final step before diving into the conclusions and results will be to do an economical viability study. The costs of producing this device will be calculated and will be compared to the current market, in order to know if the product design would not only be viable but actually profitable.

To end the project, an in-depth analysis of the results obtained, and the conclusions reached will be done. The goal of this evaluation will be to see to what degree the objectives set for this project have been met.

The next chapter will be the state of the art, where the current technologies and devices will be studied. The aim of it is to understand what has already been done, in order to create a new innovative solution that solves the gaps in the current available products. This will also help in understanding how navigational assistive aids work which will improve this project's design.

Chapter 2. STATE OF THE ART

In the past 100 years technology has evolved more than we could have ever imagined, from the creation of computers to the internet to smartphones. This technology has helped individuals in more ways than one, it has made life easier but most importantly more accessible to all, in particular to those with disabilities. Assistive devices for the blind go back thousands of years, but what the modern development, what we know today goes back to the early 20th century. This century was crucial for the development of tools that could help the blind in their everyday lives, with the evolution of the white cane and guide dogs.

In this chapter, we will dive deeper into the current state of the art by explaining it from four different main sections. To start it is crucial to study where it all started and analyze the most basic of aids, which are guide dogs and the white cane. After, the more technological devices will be discussed, which are the Electronic Travel Aids (ETAs), which have forever changed how visually impaired individuals navigate their everyday lives. Inside ETAs there are a great number of different types of aids, which is why they will be analyzed one by one. To end this chapter, a section has been dedicated specifically to the actual technological components and sensors that allow these travel aids to work.

2.1 GUIDE DOGS

Guide dogs are specially trained dogs that assist blind or visually impaired individuals to allow them to have more mobility and independence. These dogs are especially trained for the first months of their lives to learn how to navigate complex urban environments, learning how to stop at a traffic sign, stay in the pedestrian's path, avoid obstacles, cross the street safely… Overall keeping their handler safe.

Although there is evidence that guide dogs existed thousands of years ago, the modern concept of a guide dog was created during the First World War, in order to aid veterans whose vision was impaired during their service due to gas poisoning. During this time Dr. Gerhard Stalling started to experiment with the idea that dogs could help the blind soldiers and started exploring ways of properly training them. In 1916 he opened the first school for guide dogs, which trained canines not only for ex-soldiers but also to blind civilians. Although it did not last long with the school having to shut down in 1926, it had already created a worldwide movement and other schools were starting to open. Since that moment the training of dogs for the blind has become more and more popular, with more research being done about how to best train them. This has changed the lives of thousands of blind individuals who would not be able to live their normal lives without the help of their k9 friends. [5]

[5] International Guide Dog Federation (retrieved in 2024).

It is necessary to look at this because a lot of the robots being created nowadays are based on these guide dogs and how they help the visually impaired. The most important part to look at is the actual harness that the dogs must wear, since this is the tool that truly allows communication between the owner and the dog. The harness is composed of three main parts: Harness body, handle and harness straps. The most important part to study for robotics is the handle and why it is the way it is. The handle is a rigid u-shaped metal or plastic bar, the fact that it is solid allows the handler to feel the dog's movements, such as when he turns, stops or avoids an obstacle. The same can be said for the dog, it can feel if the owner wants them to turn, stop or continue straight. This may seem simple and intuitive, but it is crucial when translating it to an assistive robot, the blind individual must also feel all the subtle movements and therefore the rigidity of the bar is essential. Therefore, this simple structure is implemented in more technologically complicated assistive devices.

Figure 5: Guide dog. Source: American Kennel Club (retrieved in 2024).

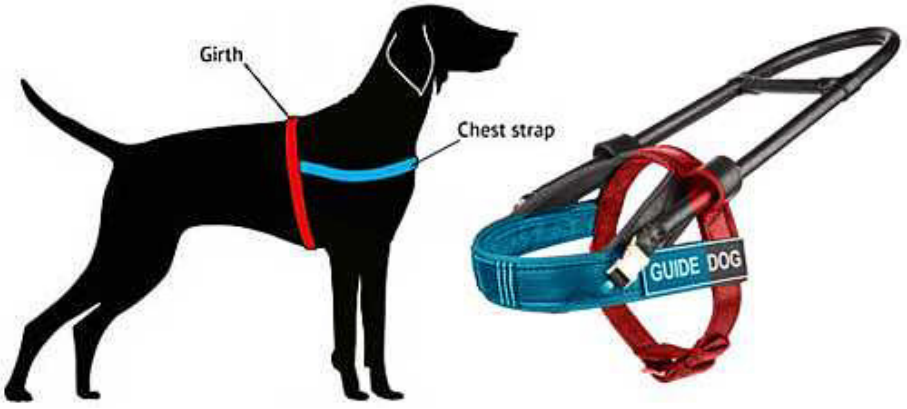


Figure 6: Guide dog harness schematic. Source: DT Dog Collars (retrieved in 2024).

2.2 WHITE CANE

The other main tool used by the blind is the traditional white cane, a mobility device that allows them to detect obstacles and navigate their surrounding more safely. Its other main purpose is to signal others that the person holding the cane is in fact blind, allowing them to have another level of security since people are aware that they cannot see. This type of has existed for thousands of years, going back to ancient times where visually impaired individuals would use staff or canes to move independently. However, the modern-day white cane did not come until the early 20th century where it was unified to be the same for all, adding the white color to signify blindness.[6]

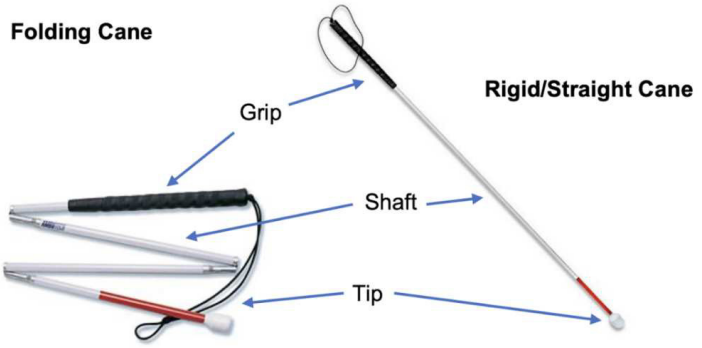


Figure 7: White cane schematic. Source: Guide Dogs Australia (retrieved in 2024.).

As with the guide dogs, the study of the white cane is crucial in order to understand the technological developments that have been made. The cane is a rigid metal or plastic, with some models being able to be folded that detects obstacles for the blind. A great deal of new assistive devices are based on this model due to the incredible simplicity but powerful use. Therefore, it is important to understand how they work, the blind individual holds the cane making a 45-degree angle with the floor and does a sweeping motion to detect obstacles ahead and to the side. This mechanism can then be translated to canes with technological advancements.

[6] Strong (2009).

2.3 ELECTRONIC TRAVEL AIDS (ETAS)

As stated before, both the guide dog and white cane are powerful tools that allow blind individuals to navigate their surroundings more safely and gain back their independence. However, both of them have significant limitations that hinder the user's safety. For example, white canes are unable to detect any obstacles above knee level and cannot provide information about the user's geospatial location. Furthermore, below knee level the detection of objects is limited to the tip of the cane, which can scan a volume of less than 1m x 1m x 0.4m at a time. [7] Similarly, guide dogs have an inability to interpret more complex urban situations, such as knowing how to interpret a traffic light, as well as being unable to communicate detailed information to their owners.

These extreme limitations have created the necessity of developing new innovative solutions to create better and safer assistive devices based on technology. These technological solutions are called Electronic Travel Aids or ETAs for short. ETAs incorporate different technologies to detect obstacles and provide tactile or auditory feedback to their user. Their goal is to guide the blind individual from point A to point B safely, as if they had no visual impediment. Their implementation has considerable improved the number of accidents blind people suffer and has also incredibly improved their independence, overall enhancing their quality of life.

Electronic travel aids can be categorized in many different ways, depending on their physical characteristics they can be divided into three: Robotic Navigation Aids, Smartphone Apps and Wearable Attachments.[8]

2.3.1 ROBOTIC NAVIGATION AIDS (RNA)

Robotic Navigation Aids are technology-based devices that assist visually impaired individuals navigate their surroundings safely. These types of ETAs include a grand variety of sensors that allow them to detect obstacles and guide the user through complex urban environments. Inside this category there are different RNAs with different characterizes and levels of guidance.

2.3.1.1 Smart Canes

The most important is the Smart Cane, this is an improved version of the traditional white cane. It includes sensors, such as ultrasonic sensos, infrared sensor, LiDAR systems or even cameras that allows it to better detect obstacles and then has tactile or auditory

[7] Di Mattia et al (2016).
[8] Romlay et al (2021).

feedback to provide that information to the user. A great advantage is that since they are in essence a white cane, they still function like one if the electronics fail which can be very reassuring for the person using it. In addition, they are very portable which again is incredibly useful, so although they are one of the technologies with less sensing capabilities, they are still very useful and therefore it is important to analyze them. In studying this type of RNAs different patents were examined and out of all of them I would like to highlight three of them.

Patent Number	**Applicant(s)**	**Inventor(s) Last Name**	**Filling Date**	**Publication Date**
US9155675	University of Arkansas	Ye	October 12th 2012	October 13th 2015
US8922759	MESA Imaging AG	Gassert, Kim, Oggier, Riesch, Deschler, Prott, Schneller and Hayward	March 22nd 2013	December 30th 2014
AU2012317177	Indian Institute of Technology	Subhashrao, Mahadevappa and Mukhopadhyay	September 28th 2012	April 4th 2013

Table 1: Smart Cane patents analyzed. Source: Own elaboration (2024).

All of these patents represent a type of smart cane and as can be seen from the table, they were publicized at similar times. Something to point out is that they are not precisely recent, being publicized around 10 years ago. This is completely logical due to the fact that the first steps in creating technology for the blind was to take the traditional systems and enhance them, in this case the traditional white cane. This is the reason why out of all of the ETAs, enhanced white canes are one of the most limited devices but that is not to say that they are not safe or useful. In this section the three patents will be further analyzed to gain an understanding of how they work and what characteristics can then be incorporated into this project.

The first patent (US9155675) is a portable robotic device that integrates a 3D imaging sensor into a white cane. In addition, it has a portable computer, a tactile device and a wireless headset. The imaging sensor, which is usually a time-of-flight camera or TOF camera, scans the environment to obtain the necessary data. The computer then process this data by implementing 3D data segmentation in real time to identify obstacles and their distances, which is then translated to a signal that the user receives through the headset or the tactile device. It also includes wayfinding, which allows the RNA to guide the

user to the destination they want to go to. Lastly, it is uses the Visual Ranging Odometry method to estimate the position and orientation of the cane which is crucial to correctly guide the individual. Although complex due to the many algorithms needed, this method is extremely interesting since it can replace the GPS in zones where the signal is weak or nonexistent.

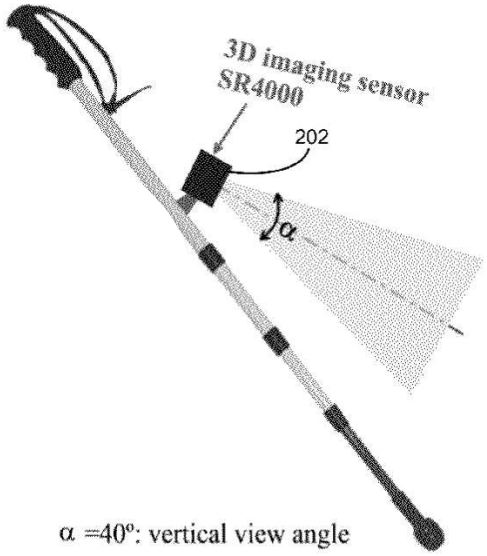


Figure 8: White cane with TOF camera.

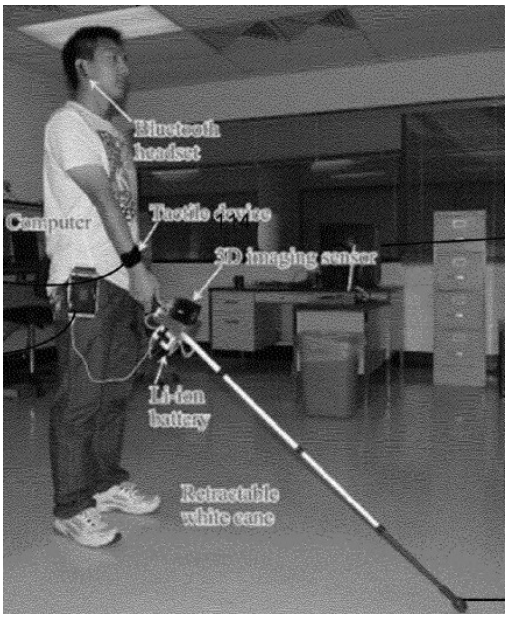


Figure 9: Complete white cane with TOF camera.

The second patent (US8922759) is very similar to the first, also integrating a time-of-flight camera into a white cane. The main difference is that this time it integrates all of the necessary sensor into the actual white cane without the need of carrying extra devices which can be an improvement on the previous. The other big difference is that it includes other sensor such as gyroscopes, accelerometers, GPS and compasses which allow the device to know the orientation and motion of the cane, which can then be processed by the evaluation unit to guide the user. It also allows for a mode selection, depending on the needs of the blind individual in each moment, for example a walking mode and a scanning mode. All of these features make this enhanced cane compact and user friendly.

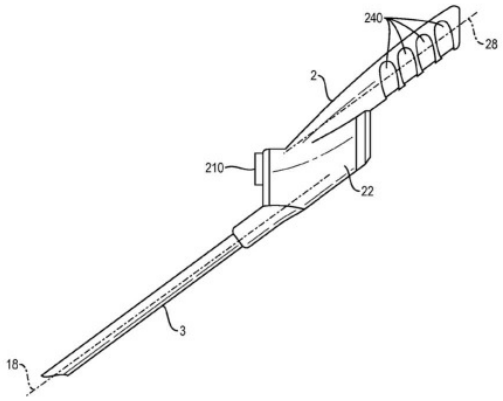


Figure 10: Enhanced white cane

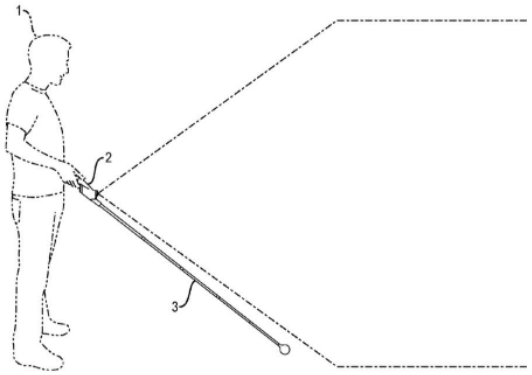


Figure 11: Person holding an enhanced white cane.

The last patent (AU2012317177) analyzed is a bit different to the previous ones, due to the fact that it does not work with a TOF camera but with ultrasonic sensors. It has eight ultrasonic sensors that are strategically placed to detect obstacle ahead, above knee level and below. It also includes specialized sensors to detect wet floors, metal floor or even fire, this is called the danger unit and can be incredibly useful. This cane also has a special focus on the detection of stairs, positioning three front sensors to identify upwards stairs and two

back sensors to detect downward stairs. To then transmit all of this information to the user the enhanced cane uses prerecorded auditory, with more imminent alerts being transmitted through a buzzer. The main advantage is the simplicity of it, that allows for a light weight, compact and user-friendly device. The issue is that ultrasonic sensors are a lot less accurate than TOF cameras and therefore the obstacle detection can fail, especially the calculation of distance to the object. In addition, this device does not have a way to guide the user from point A to point B since it does not have a GPS or wayfinding method. There are still important factors to take away, that sometimes simplicity is sufficient and safe.

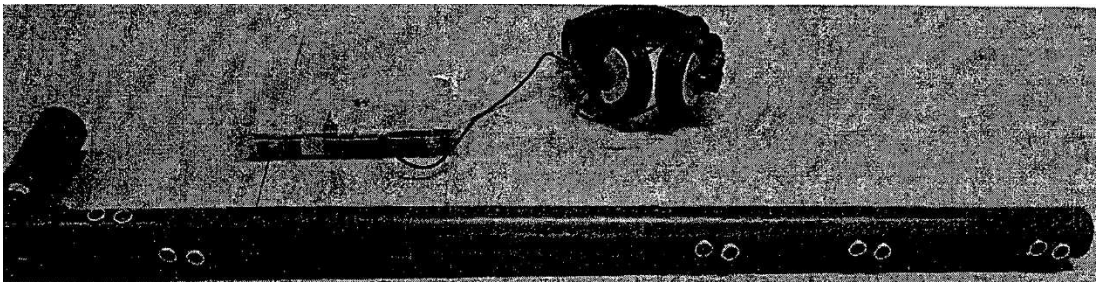


Figure 12: VENUCANE device.

After this overview of the three patents, it can be seen that all of them have similar features that clearly enhance the capabilities of a traditional white cane. Although they still face some challenges such as cost, complexity, power consumption or inaccuracy in some cases, a lot can be taken away for the development of this project.

2.3.1.2 Robotic Guide Dog

When we think of robotic guide dogs with of exactly what the name indicates, a copy of traditional guide dogs in robotic form. This is an emerging technology and is a lot more recent than the smart canes. These systems have very advanced sensors, such as cameras and LiDAR sensors, and they also use artificial intelligence and machine learning to enhance their capabilities. They are able to navigate very complex urban environments and take the user from point A to point B safely, since they incorporate navigation capabilities. Mechanically they are extremely complex, due to the fact that they must be able to walk through different terrains as well as go up and down stairs. This makes them extremely safe for the visually impaired, with them being able to map all of their surrounding and tailor that information to the needs of each user. Now, clearly as for safety and accuracy goes robotic guide dogs take the crown but with all of this, problems arise. Firstly, all of these features make these robotic navigation aids extremely complex, which in turn means that they are highly expensive making them not accessible to a wide section of the population. Another thing that goes hand I hand with complexity is maintenance, the more technology the more issues the robot may have, if any of the sensors or systems fail it can cause an extreme safety risk. In addition, they require a lot of processing power which means they use very heavy and large hardware making them not portable and very difficult to manage. Not only

does these make them not portable but it also causes an issue for battery life, which may be a lot shorter than other simpler RNAs.

As for current market availability, there isn't a lot which goes to show the long way this technology has to go. It is necessary for the state of the art to evolve to a point where the choice between extreme complexity and safety doesn't have to be made. It does not seem like real robot dogs are the immediate solution for assisting the blind but with further investigation and resources they may be. Although not currently being used to aid the blind the most promising is the Unitree Go1 robot, which is already commercially available for other purposes.

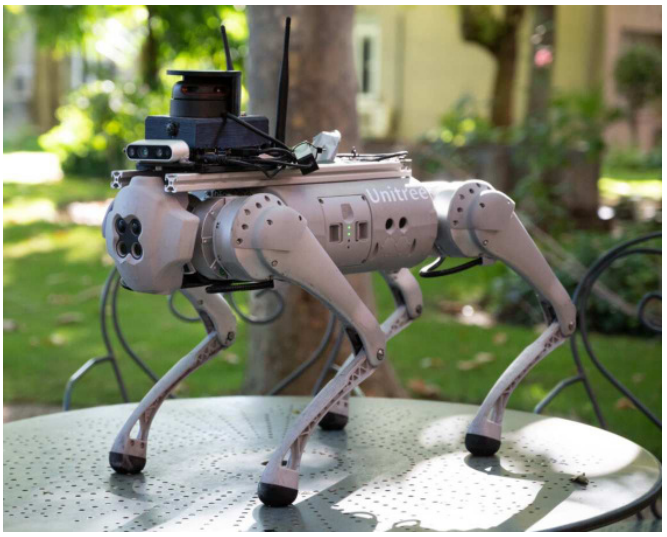


Figure 13: Unitree Go1 robot. Source: Xataka (2022).

All of these facts point at the fact that the current best solution for an ETA is not to replicate real-life dogs but to develop assistive aids in another manner. For this reason, it is necessary to look at another type of ETA that simulates these same functions but does not physically resemble dogs. These physically different robots have had a lot more success, especially in terms of market availability, that is not to say that actual guide dogs don't have a future in helping visually impaired individuals. There are two devices that must be highlighted in this section: Glide and Lysa.

Both of these robotic navigation aids are very similar not only to themselves but to the previously discussed guide dogs. They also offer navigation assistance to blind individuals using highly accurate sensors, artificial intelligence and machine learning, as well as auditory and tactile feedback for the user. The main difference with the others is their physical structure, which is composed of a rigid handle attached to a rectangular box with wheels. The wheels allow for these two systems to move with a lot more ease than actual robot guide dogs, which allows for a more reliable and user-friendly device.

The Glide is extremely portable and uses autonomous vehicle technology to guide the user. It also has two modes, one designed to follow a straight path while avoiding objects and the other to offer precise directions to a destination. The main problem, as with a lot of new and innovative technology is the cost, which will prohibit a lot of the blind population from obtaining it. Another big issue is battery life, although it is not precisely stated it is estimated to be a couple of hours for one charge, which clearly possess a problem for an individual who would like to solely relay on assistive technology during their day to day.

As for the Lysa robot, although less portable this is due to the fact that it possesses more sensors, such as LiDAR and 3D cameras which makes it the more accurate choice. It has a cost of around $3,000, which may seem highly expensive but in the world of assistive devices for the blind this price is affordable, with actual guide dogs costing between $25,000 and $60,000. But not everything is an advantage, this ETA is designed for indoor use which seriously reduces its usability and puts in question its usefulness. Finally, it is compact but a lot bulkier than the Glide which may cause difficulties transporting and using in tight spaces.

Figure 14: Glide device. Source: Glidance (retrieved in 2024).

Figure 15: Lysa device. Source: Revista Pesquisa FAPESP (2022).

Both Glide and Lysa have been an incredible advancement in the field of assistive devices but there is still a long way to go.

2.3.2 SMARTPHONE APPS

As of 2024 there are 4.88 billion smartphone users in the world, which makes up for 60.42% of the global population. Furthermore, it is estimated that by 2029 this number will grow to be 6.38 billion smartphone users [9]. Why are these figures important? Well, there are hundreds of apps destined to help the blind not only in navigating their surroundings safely but in performing everyday tasks that they would otherwise be unable to do. The rectangle that almost all of us carry in our pockets has the ability to reach billions of people and help them, that is why thousand apps for the blind have been developed and it is necessary to look at the most popular. The applications analyzed here are those designed to guide the visually impaired individual from point A to point B safely, especially in complex environments, they are called Wayfinding Apps. Out of the thousands of existing apps the two most popular for outdoor navigation are BlindSquare and Lazarillo, thus they will be the main focus of this section. [10]

[9] BankMyCell (retrieved in 2024)
[10] Theodorou, Tsiligkos, & Meliones (2023).

2.3.2.1 BlindSquare

BlindSquare is an app designed to help the blind navigate outdoor and indoor environments. It works in a very user-friendly way; the person selects their preferred destination via voice commands and the application guides the user to it. In order to do so, it uses a combination of GPS technology for outdoors and BLE beacons for indoors to know the user's exact position and guide them accordingly. Moreover, it is completely customizable, from the interface to the feedback system which is a great way to make the individuals feel safer and make less mistakes when following inputting and following instructions. It also uses other platforms such as Foursquare and OpenStreetMap, this allows the app to work in even more environments but does come with issues. The main issue is the fact that it heavily relays on external services, which means that when they are not reachable the application inevitably fails and can cause serious danger to the user. There are also other areas in which the device fails to provide sufficient information, such as telling the user what the objects actually are to avoid injuries, like staircases, benches or bollards.

2.3.2.2 Lazarillo

Lazarillo is a similar application that helps user navigate both outdoor and indoor environments. It also uses GPS technology and BLE beacons to properly guide the user through voice commands. Similarly to the previous it can be customized to adjust language, alerts and other instructions. The main advantage of this specific app is its integration with public transport which adds another layer of aid that can be extremely helpful, especially when navigating large urban cities.

Now that this brief overview has been done it is important to compare both applications. The big difference is that BlindSquare has a monthly cost of $40 which although very expensive for a mobile app very affordable for an assistive device. In addition, it is only available for iPhone's making it less accessible to all. These restrictions may seem extreme, but the cost comes due to the integration with Foursquare which makes this application a lot more accurate and reliable than Lazarillo, although this last one comes on top for public transportation. In conclusion both have their advantages and disadvantages and although they may have less power than actual robotic navigation aids, they are an incredible useful tool, especially when paired with a white cane or even a guide dog. They still have quite a way to go in order for them to be used by themselves but as of today they have already completely transformed the lives of thousands of blind individuals due to their affordability and accessibility.

2.3.3 WEARABLE DEVICES

These devices are designed to improve the user's mobility while remaining discreet and hands-free. They can take many different forms, such as glasses, gloves, vests, wristbands, belts, earpieces or headgear. In 2023 an in-depth study was done about the different wearable devices and what body parts they were most commonly worn on.

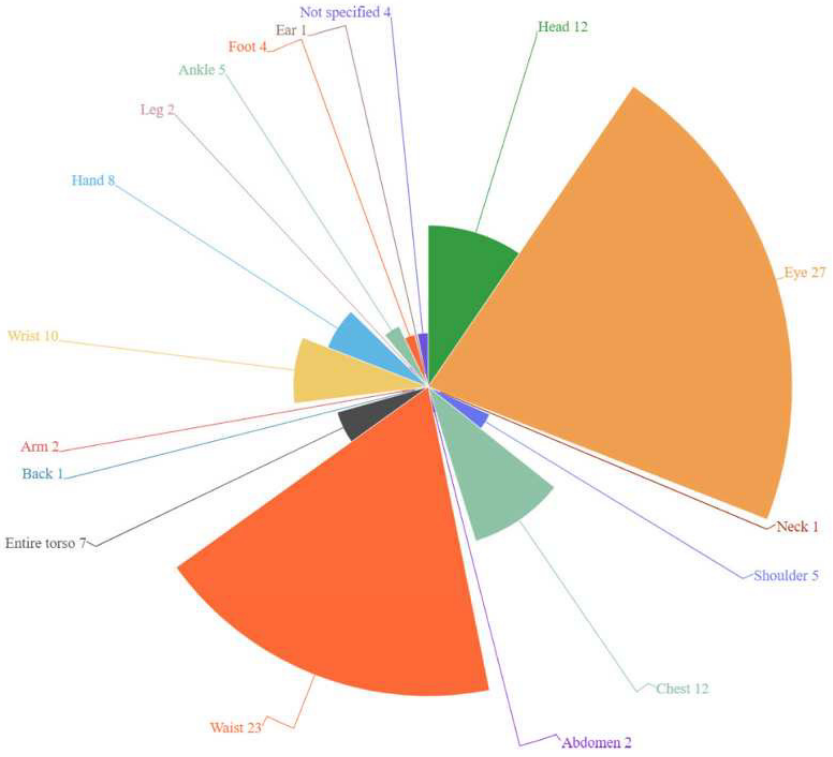


Figure 16: Number of body parts to be worn. Source: Xu et al (2023).

As seen in the previous figure there is a huge variety of wearable devices, thus not all of them can be studied in depth. For this section and to further help the development of this project different patents were analyzed and the most relevant were chosen to include in the paper.

Wearable Type	Patent Number	Applicant(s)	Inventor(s) Last Name	Filling Date	Publication Date
Bracelet	US9990860B2	BOE Technology Group	Zhang	July 15th 2015	Jun 5th 2018
Clip	US10024678B2	Toyota Motor Engineering & Manufacturing Nort America	Moore, Djugash and Ota	September 17th 2014	July 17th 2018
Earpiece	US10024667B2	Toyota Motor Engineering & Manufacturing Nort America	Moore, Djugash and Ota	August 1st 2014	July 17th 2018
Glasses	US11852493B1	Vortant Technologies	Schaefer	July 15th 2021	December 26th 2023
	US9922236B2	Toyota Motor Engineering & Manufacturing Nort America	Moore, Djugash and Ota	September 17th 2014	March 20th 2018
	US9508269B2	Echo-Sense INC	Slamka	November 16th 2011	November 29th 2016
Headgear	US10535280B2	Jacob Kohn	Kohn	January 20th 2017	January 14th 2020
	US10528815B2	VasuYantra Corp	Maheriya and Srivastava	October 9th 2017	January 7th 2020
Necklace	US9915545B2	Toyota Motor Engineering & Manufacturing Nort America	Chen, Djugash and Yamamoto	December 5th 2014	March 13th 2018

Table 2: Wearable devices patents. Source: Own elaboration (2024).

Although there are many more types of wearable device that the ones exposed in the previous table, all of these patents have characteristics that make them very interesting when studying the current state of the art. They main advantage of this type of ETA is not only their portability due to them being attached to your body but the fact that they allow the user to be hands free. Not only is it more comfortable to be able to use your hands but it is also a lot safer, for example in the case the individual falls down they can stop the fall with their hands. In addition, depending on the model they can more discrete than other types of assistive aids, in the case of a simple clip it can be a lot less noticeable than for example the Glide, which for some can be a determining fact. But not everything is an advantage, the fact that has to be worn although allows to free the hands can become uncomfortable when worn during long periods. A great example would be an earpiece that could create soreness if worn a whole day. Additionally, these devices aim to be small and lightweight in order for the user to be able to wear them, but this means they don't have as many sensors or complex system as other ETAs. This fact causes wearable devices to focus not on guiding the visually impaired individual but to assist him, which means that most of time the device does not stand on its own. Individuals who use these devices on a daily basis pair them with other assistive devices, such as a white cane.

2.4 TECHNOLOGICAL COMPONENTS

Now that an overview of the different types of assistive devices for the blind has been done it is necessary to do a more in-depth study of the actual sensors and systems that they use. Each of the components of an electronic travel aid is aimed to aid the user in a specific area: Obstacle detection, geolocation and navigation and user feedback. It must be noted that in order for ETAs to provide the user with the best possible assistance and safely guide them individual they use more than one technology.

2.4.1 OBSTACLE DETECTION

This section consists of a brief overview of the obstacle detection sensors that can be implemented in this particular ETA taking into consideration the objectives set. There are a great number of sensors available, each with advantages and disadvantages. The most common sensors are: Ultrasonic, infrared, LiDAR, camera-based and radar.

Although radar sensors have a great deal of advantages that makes them be one of the most accurate, they are not suitable for this specific project. The main reason is their high complexity, that results in heavier and larger devices which directly contradicts the objective of creating a lightweight and compact ETA. Additionally, they are very costly, due to these reasons they will be left out of this analysis. Therefore, the research will only be focused on ultrasonic, infrared, LiDAR and camera-based sensors. This ensures that the ETA is lightweight and relatively affordable.

2.4.1.1 Ultrasonic Sensors

One of the most used technologies in ETAs are ultrasonic sensors, because they can detect and measure the distance to an object without making physical contact. These sensors emit sound waves at a very high frequency (above 20kHz) that bounce back to them after they hit an object. They then measure the time of flight (time it took for the waves to return to the sensor) which allows them to not only detect the presence of an object but to calculate how far away it is. Therefore, this technology acts as an ultrasonic wave emitter, as well as an ultrasonic wave receiver. The calculation is possible because the speed of sound is a known constant that only varies with temperature (usually 343 m/s) and the only unknown quantity is the distance. The fact that the temperature varies is a source of concern that must be taken into account in order to have the most accurate measurements possible. The distance is determined using the following formula: [11]

$$Distance\ to\ Object = \frac{Total\ Time\ of\ Flight\ \times\ Speed\ of\ Sound}{2} \quad (1)$$

t must be noted that the distance returned by the ultrasonic sensor will be the closest distance to the device, therefore it only returns one measurement at a time. The following figure shows a simple drawing of how the emission and reflected wave works.

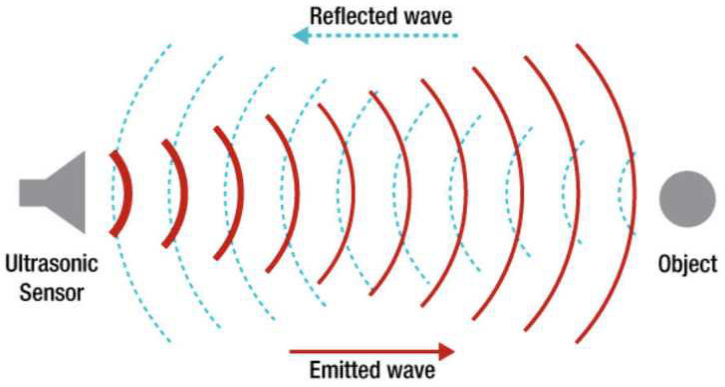


Figure 17: Representation of ultrasonic sensor. Source: Morgan (2014).

The issue with this is that it is possible for the sensor to mistake one object for another due to unintended echoes, that would cause the system to think, and object is closer than in reality.

[11] Toa & Whitehead (2020).

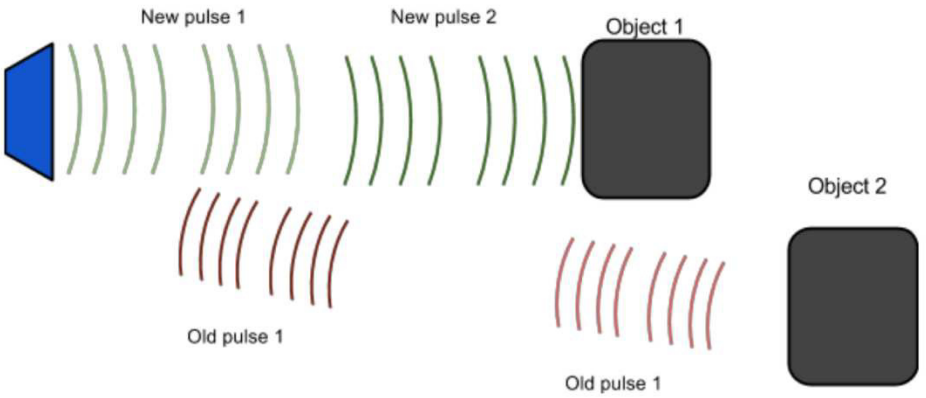


Figure 18: Representation of ultrasonic sensor failing. Source: Morgan (2014).

In the figure above, the echo of the new pulse 1 will be confused by the return of the old pulse 1, making it seem as if object 1 is a lot closer.

Ultrasonic sensors have a great deal of advantages, for example they work with a wide range of materials and conditions, like poor lighting but still have trouble when it comes to reflective surfaces or irregularly shaped objects which may reflect the waves in a weird manner [12]. They have other serious limitations that can put at risk the safety of the visually impaired individual. For example, the speed of sound has is not as fast as the speed of light, which mean that for objects that are further away the detection may not be as immediate as one would hope. Another issue found with this technology is the limited range and angle. The range depends mainly on the objects size and shape [13] and the specific sensor used. The most common sensor used for robotics is the HC-SR04 Ultrasonic Sensor due to its cost, size, weight and ease of implementations. This sensor has a measuring angle of 30 degrees and a distance range of 2 to 400 cm [14]. These restricted limitations mean that more than one ultrasonic sensor would have to be used and they would possibly have to be paired with other more powerful technologies.

[12] Mocanu, Tapu, & Zaharia (2016).
[13] Morgan (2014).
[14] Ajmera (2017).

2.4.1.2 Infrared Sensors

Infrared sensors follow the same principal as ultrasonic sensor, they emit an infrared light, which then reflects from the surface of an object and goes back to the sensor. Similarly to ultrasonic sensors to know where the item is, they can measure the time it takes for the beam to bounce, but they can also work by measuring the intensity or the angle of the reflected light. Therefore, they work as an emitter and as a receiver.[15]

They have advantages, such as their small size, low cost and stable performance but they have some serious issues that must be taken into account. They have problems when being exposed to sunlight and other sources that emit IR waves, this is because the receiver considers them a signal which interferes with proper object detection. In addition, surfaces that are extremely dark or absorbent tend to not reflect the light and therefore the sensor is unable to detect it. Lastly, the range and angle of detection must be considered and although it depends on the exact model it tends to be small. The angle is not that big of an issue since it can easily be solved due to its fast actuation time, a servo motor can be implemented to rotate it in a sweeping motion and more sensors can be used.

The most popular are the Sharp Infrared Sensors, since they are very powerful while still being cheap, small and having low power consumption. They use the triangle principle, shown in the figure below.

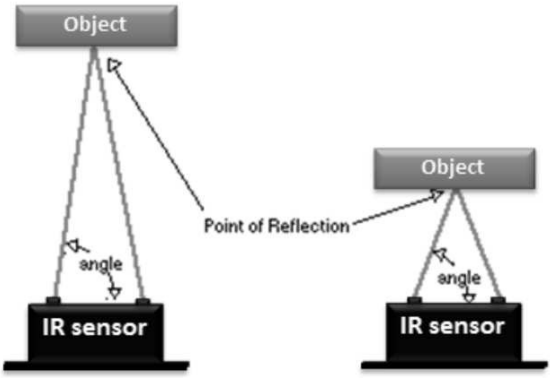


Figure 19: Representation of infrared sensor.

[15] Rashid & Ali (2018).

This means that the sensor does not have a linear output, which causes a problem when trying to detect objects that are at a very close range [16]. The models that could work for this project are:

- **Sharp GP2Y0A02YK0F:** Range of 20 cm to 150 cm. [17]

- **Sharp GP2Y0A710K0F**: Range of 100 cm to 550 cm. [18]

Due to these ranges both sensors would have to be used together but there would still be an issue for very close objects. Therefore, IR sensors could be used and might be necessary, but they would have to be coupled with another type of technology.

2.4.1.3 Light Detection and Ranging (LiDAR)

LiDAR technology is among the most popular sensor for obstacle detection, and it is widely used in self-driving cars due to its ability to scan its field of view and create high-resolution 3D maps. These sensors operate similarly to ultrasonic sensors, by emitting laser beams and having them reflect off an object and return. However, LiDAR is much faster and more accurate because it uses the speed of light instead of the speed of sound [19]. Therefore, the simplified equation to calculate distance is:

$$Distance\ to\ Object = \frac{Total\ Time\ of\ Flight\ \times\ Speed\ of\ Light}{2} \quad (2)$$

They calculate the distance to an object by measuring time of flight but to do this as accurately as possible they also take into account the intensity, phase and frequency of the beams. In order for this to work, the sensors have a laser transmitter, a photodetector receiver and a beam steering device.

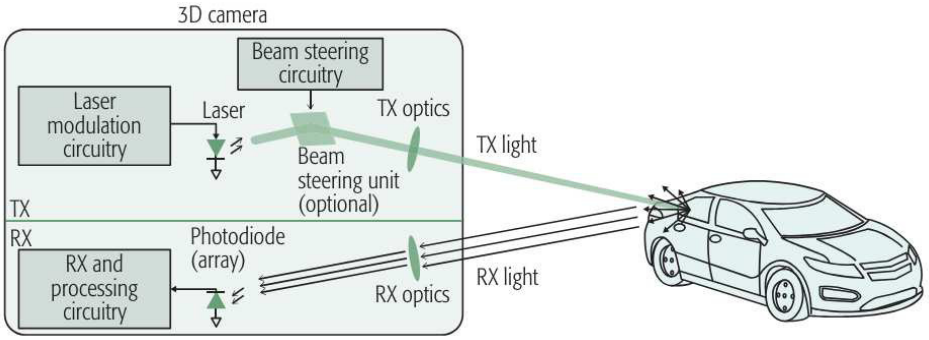


Figure 20: Representation of LiDAR. Source: Behroozpour et al (2017).

[16] Sharp Corporation A (2006).
[17] Sharp Corporation B (2006).
[18] Li & Ibanez-Guzman (2020).
[19] Behroozpour et al (2017).

As for the range, it is estimated based on the time delay and variations in the energy that's reflected, but it is very long. Additionally, it has a 360-degree angle of detection, which ensures that there are no blind spots. All of these characteristics make LiDAR one of the best and most precise technologies for obstacle detection, but it still has disadvantages [20]. These sensors are extremely expensive when compared to others and they also have a higher power consumption which could cause problems. They also have issues in certain weather conditions, such as heavy rain or fog.

For the very same reason that camera-based and radar sensors were left out of this analysis, the 3D LiDAR sensors are too expensive to implement, with the cheapest being at around $500. Therefore, if this technology were to be used, a 2D sensor would be more appropriate for the requirements of this project. Some of the most cost-effective are the RPLIDAR Sensors and there is one model that is particularly interesting:

- **RPLIDAR A1:**
    - Range of 0.15 - 6 meters.
    - Cost of around $100. [21]
    - Angle of 360 degrees.

2.4.1.4 Camera-based Sensors

Camera-based sensors are very popular due to the fact that they are extremely precise, since they are capable of duplicating human vision. They take pictures or videos of their environment and the process the data using computer vision algorithms to identify and classify objects, as well as detect motion, distance to them and their depth. The great advantage of this technology is that it can use machine learning and other algorithms to not only detect that there is a sign, or a traffic light ahead but actually know what the sign says or whether or not the traffic light is red or green. The fact that they use machine learning ensures that the results are reliable since the images are enhanced and noise is removed when they are being preprocessed.

Although they have a great deal of advantages, they still have problems that need to be taken into account. They have trouble when there is poor lighting due to the fact that the images show up too dark to recognize anything, this happens at night but also when there are bad weather conditions. Even with preprocessing in a lot of cases the sensor fails to detect objects correctly. Additionally, since this requires a lot of computational power which cause two issues: Delay in the image processing which can cause accidents and a lot of power consumption. The last can be solved by adding code that allows the camera to only capturing points of interest [22]. It must also be taken into account the complexity that

[20] Ahmad et al. (2013).
[21] SLAMTEC (2017).
[22] Mukhiddinov & Cho (2021).

comes with using this type of sensor, as well as its cost. The cost can vary a lot depending on the model, the cheaper the less processing power, therefore if this technology were to be used a balance would have to be found. One of the most balanced models out there is the Raspberry Pi Camera Module V2, it is cheap but can still 1080p quality video.

2.4.2 GEOLOCATION AND NAVIGATION

The next objective that must be reached is the device's capability of knowing not only were the user is but were they are facing. For this to be a reality different sensor will have to be analyzed and implemented. The most helpful technologies for this are: GPS, inertial measurement unit (IMU), LiDAR and camera-based sensors. In the previous section LiDAR and camera-based sensors were already discussed and since they not only aid in obstacle detection, but also help map the environment they start to look like a good option to implement. For this reason, in the next pages only GPS and IMU will be studied. For the ETA to be able to guide the blind individual from point A to point B through the best path no extra sensors are needed and will be done using code.

2.4.2.1 GPS (Global Positioning System)

It is essential that this ETA has a GPS, a satellite-based navigation system that tracks longitude, latitude and altitude, in other words know the exact coordinates of the visually impaired individual.

As for how it works, the GPS receiver detects the signals transmitted by a network of 31 satellites. These signals contain the position of the satellite that transmitted them, and they travel at a constant speed, the speed of light. The GPS receiver can then calculate the time delay between when the signal was sent and when it was received and therefore the device's location. To properly function the signals of at least four satellites, have to reach the device in order for it to be able to triangulate to determine the exact position. If more signals are received the more accurate the location. GPS is extremely accurate, works reliable anywhere in the world and is provides real time tracking, since it is continuously updating. The one issue comes when the signals are obstructed, for example by trees, building or other structures which can lead to an unknown location. [23]

[23] Dhod et al (2017).

2.4.2.2 Inertial Measurement Unit (IMU)

The ETA must be able to know where the user is facing to guide them properly, this can be achieved by implementing an Inertial Measurement Unit (IMU). These units can measure velocity, orientation, and sometimes the gravitational force. They consist mainly of two different systems, accelerometers, and gyroscopes, with a lot of models also including magnetometers. These last ones are included due to the fact that they helps correct the gyroscopic drift that inevitably appears and thus results in more reliable data. The figure below shows a visual representation of what each of the systems does and how they work together. [24]

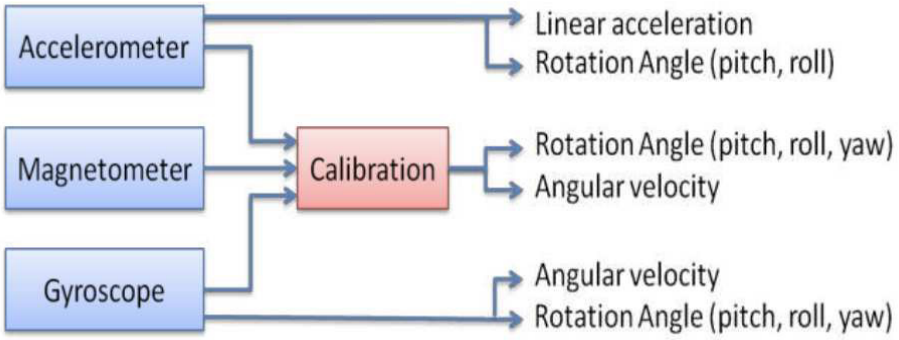


Figure 21: IMU schematic based on three sensors. Source: Ahmad et al (2013).

The accelerometer, as the name says, measures inertial acceleration, usually in three axes: x-axis, y-axis and z-axis. The most common technology used for them are the micro-electromechanical systems, also called MEMS for short. This refers to how they are manufactured, which is using silicon micro-fabrication technology that allows the creation of incredibly small and low-cost chips. Within MEMS, accelerometers can be divided depending on the actual sensing technology:

- **Capacitive:** These sensors have a mass attached to a spring and when the IMU accelerates this mass moves and changes the capacitance. This change is proportional to the acceleration and that is how it is calculated.

- **Piezoelectric:** These sensors use microscopic crystal structures that generate an electric charge when they are exposed to mechanical stress. This generated voltage is proportional to the acceleration; therefore, this one can be calculated.

[24] Ahmad et al (2013).

- **Piezoresistive:** These sensors detect the changes in electrical resistance that occurs when they are exposed to mechanical stress. This resistance is related to the accelerations and therefore this one can be calculated. [25]

The gyroscope measures angular rotation, usually in three axes: x-axis, y-axis and z-axis. Similarly to accelerometers the most used technology is MEMS, these gyroscopes have a vibrating mechanical element that allows the device to detect angular velocity. This is possible thanks to the Coriolis acceleration, which only appears when there is rotation. This causes the transfer of energy between two vibration modes, the first being the vibrating mechanical element and the second being the vibration induced by the Coriolis effect. This transfer is detected by the gyroscope (using the same sensing devices as the accelerometer) and since the second vibration is proportional to the angular velocity this one can be calculated. [26]

The magnetometer measures magnetic field, usually in three axes: x-axis, y-axis and z-axis. Being able to detect this, allows the device to know its orientation relative to the earth's magnetic field. There are different types of magnetometers depending on the sensor used, the most popular are:

- **Hall:** These sensors consist of a conductive material and when it exposed to a magnetic field a Hall voltage is generated. This voltage difference created is proportional to the magnetic field and therefore when measured the field can be calculated.

- **Magnetoresistive:** These sensors, as the name indicates, are made out of materials that have an extreme change in resistance when exposed to magnetic fields. This change is measured with the voltage drop caused due to the presence of the field. This voltage is proportional to the magnetic field and therefore this one can be calculated.

- **Fluxgate:** These sensors have one or more ferromagnetic cores wrapped with coils of wires that are magnetized and demagnetized, these changes are measured which is how they detect magnetic fields. [27]

[25] Andrejašic (2008).
[26] Passaro et al (2017).
[27] Li & Wang (2014).

2.4.3 USER FEEDBACK

It is crucial that this ETA has some type of feedback so that the user can be aware of what is happening in their surroundings. As stated in the previous section, this feedback can be provided using two different modes: Auditory and tactile.

- Auditory Feedback: This mode is very effective to communicate very detailed information, through the use of spoken directions or other nonverbal sounds. For example, a beep might indicate an obstacle, while a voice command could say, “Obstacle ahead”. The issue comes when it is used in loud environments where the user might not hear the auditory signal or distinguish its meaning.

- Tactile Feedback: This mode usually consists of different vibrations, where depending on their pattern it communicates one thing or another. For example, a long vibration could mean there is an obstacle ahead, while consecutive short vibrations might indicate that the user needs to turn left. While very useful in noisy environments, it falls short when there is a need to provide very detailed information.

Clearly, the most effective approach will be a combination of both forms of feedback so that the communication can be tailored to the circumstances and the individual’s needs. Detailed information can be provided through auditory cues, while other alerts can be conveyed using tactile feedback. The fusion of the two ensures that the user receives the best guidance for each particular situation.

2.5 CONCLUSIONS

In this chapter the current state of the art has been analyzed, mainly focusing on the different types of ETAs and the technologies that form them. With this study several conclusions have been reached

- Each type of ETAs has severe faults that should be addressed. Enhanced white canes are too simple, with a lot of crucial functions unable to be met in the great majority of cases. For example, they can’t detect overhead obstacles or guide the user. Wearable devices, as well as smartphone apps are not powerful enough to be used on their own, meaning that they would have to be paired with a white cane or guide dog. Robotic guide dogs are in many ways too elaborate, which enhances the chances of an issue and makes them expensive and not portable. Therefore, to achieve an optimal travel aid, a device must be created that combines the best characteristics of each of them, such as obstacle detection, including overhead, user

guidance, portability, low-cost, and most importantly be capable of standing on its own.

- For the sensors, it was found that the most used are: Ultrasonic sensors, infrared sensors, LiDAR and cameras. After analyzing each of them it was found that infrared and ultrasonic sensors are pretty much used for the same purposes and since ultrasonic are known to be more accurate and reliable, it must be the on implemented. As for the LiDAR, two types were studied 2D and 3D. It was concluded that the capabilities of the 2D LiDAR were more than enough for mapping and obstacle detection, and it had the advantage of less processing power needed. Lastly, for the camera-based sensors it was found that they add a layer of complication to any device they are implemented on, and therefore, its functions should be kept to a minimum.

- For the other technological components observed, it was concluded that all of them serve a significant and useful purpose to a travel aid.

Chapter 3. DEVICE DESIGN

In the previous chapter the state of the art of the current assistive devices for the blind was analyzed. This was key in designing this guidance system. This chapter will focus on the conceptual design, developing the theoretical model which will then be programmed and simulated. It will be divided into five sections, the first being the system definition where the overall function will be explained, as well as the modules that form the ETA and how they communicate with one another. The next section will be the physical design, and then the development of the technological components, which will dive deeper into where to position them, why and how they work. Finally, this chapter will end with an explanation of the path guiding and obstacle avoidance algorithm which will allow the ETA to achieve its key objective, guide the visually impaired individual safely.

3.1 SYSTEM DEFINITION

In this first section of the chapter, the overall system definition for the ETA will be done, by explaining its function on a high level. To start, the following flow chart has been created to show how each of the systems signals interact and communicate with one ano. ther.

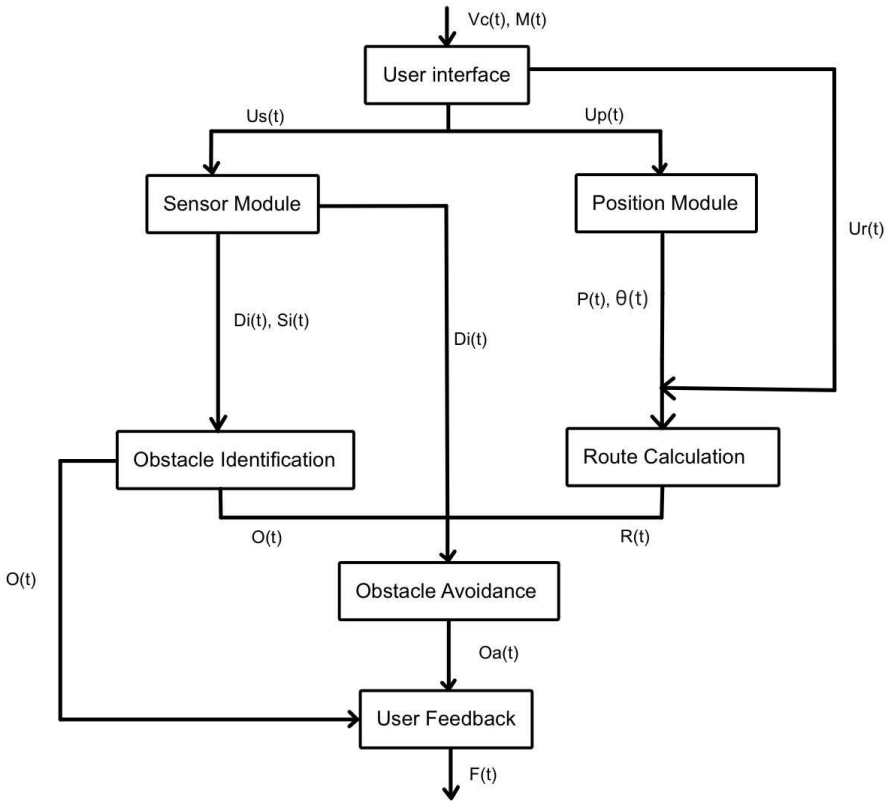


Figure 22: System flow chart. Source: Own elaboration (2024).

The system starts with the signals inputted by the user into the user interface, which can be transmitted manually or via a voice command. These signals are then processed by the user interface and sent to the sensor module, the position module and the route calculation block.

$$Us(t) = f_{User\ Interface,Us}(Vc(t), M(t)) \quad (3)$$
$$Up(t) = f_{User\ Interface,Up}\big(Vc(t), M(t)\big) \quad (4)$$
$$Ur(t) = f_{User\ Interface,Ur}(Vc(t), M(t)) \quad (5)$$

These signals will contain very simple information, with $Us(t)$ and $Up(t)$ only transmitting on and off information and $Ur(t)$ the user's desired destination.

The sensor module calculates the distance from the user to an obstacle and then sends that information to the obstacle identification and obstacle avoidance blocks. It will also have another signal outputted which will send out any other sensor information that is needed in order to identify and avoid an obstacle, this is especially added for a camera-based sensor.

$$D_i(t) = f_{Sensor\ Module,D}(Us(t)) \quad (6)$$
$$S_i(t) = f_{Sensor\ Module,S}(Us(t)) \quad (7)$$

The position module calculates the user's position and orientation, and then transmits it to the route calculation block.

$$P(t) = f_{Position\ Module,P}(Up(t)) \quad (8)$$
$$\theta(t) = f_{Position\ Module,\theta}(Up(t)) \quad (9)$$

The route calculation block does exactly what its name says, calculates the best path from the user's location to their desired one.

$$R(t) = f_{Route\ Calculation}\,(Ur(t), P(t), \theta(t)) \quad (10)$$

The obstacle identification block receives a signal with the distance measured by each of the sensors and another one with other relevant information. These distances allow the device to know whether or not an object is present (if the distance is shorter than the sensors range there is an object). If an obstacle is detected the block will categorize it into a staircase, a traffic light or any other object. This information will be sent to the user feedback module, so that it can then be transmitted to the user to make them aware of their surroundings and to the obstacle avoidance block, since depending on the type of obstacle the device will have to react one way or another.

$$O(t) = f_{Obstacle\ Identification}(D_i(t), S_i(t)) \quad (11)$$

The obstacle avoidance block will receive the distance to the object, the type of object and the route path. With this information it will calculate and execute the best path to move forward safely, whether that be to turn, stop or continue but always taking into account the user's desired final destination. The output of this block will be a signal to the user feedback module, so that it can later be transmitted to the user.

$$Oa(t) = f_{Obstacle\ Avoidance}(O(t), D_i(t), R(t)) \qquad (12)$$

The final block is the user feedback module, which is in charge of communicating to the visually impaired individual the necessary information via a speaker. It will receive the type of obstacle detected, as well as the movement command executed by the obstacle avoidance block. It will process this information and communicate it to the user if pertinent, meaning if it is determined that no change in path is needed then no command would be given to the user.

$$F(t) = f_{User\ Feedback}(O(t), D_i(t), R(t)) \qquad (13)$$

3.2 PHYSICAL DESIGN

This sections aim is to arrive at a physical model of the travel aid before taking into account the sensors and electronical components that will be included. This ETA will be composed of two main parts: a wheel and a bar. This was the final model chosen for the following reasons:

1. Guiding: This travel aids main aim is to actually guide the blind individual from point A to point B, therefore something must 'pull' or 'move' them towards a specific direction. The best way to do so, is by having a wheel with a motor and a steering actuator that can lead the user, by moving, turning and stopping.

2. User-friendly: Having a wheel guiding the user allows this system to be very intuitive to use, as well as have a short learning curve. The user must just let the device lead them towards the correct direction making it very user-friendly. In addition, the decision to have the wheel connected to a rigid be as simple as possible, which shortens the learning curve even further.

3.2.1 WHEEL

One of the bigger issues this design has is the detection of stairs, as well as being able to efficiently go up and down them. Due to its complexity and interest in this project stair detection will have its own section in this chapter. But before being able to detect stairs it is crucial to have a device that can actually navigate them. Furthermore, the wheel design will define where the sensors can and should be to not only detect stairs but all obstacles.

Our objectives for the stairs are very simple although complex to achieve. The electronic travel aid must detect up and down stairs, as well as be able to navigate them. After analyzing different patents and scholar documents, I found there was a lack of research and ideas when it comes to a single wheel navigating stairs, the research focusses on the use of at least two wheels. This is primarily due to the fact that two wheels allow the device to be more stable and have more power to climb the stairs, the big issue is that one of the main objectives of this project is to design a portable and lightweight system. The use of two wheels would interfere with the achievement of this goal and therefore it has been concluded that the design must only have one wheel. Although during the research done only two or more wheeled stair climbing vehicles were found, one of the was particularly interesting and can be adapted to use only one wheel, for this reason this section will be focused on it. The main purpose is to analyze this patent and implement the necessary changes to using in our ETA as a one-wheel solution.

Patent Number	**Applicant(s)**	**Inventor(s) Last Name**	**Filling Date**	**Publication Date**
20110127732	Individual	Mann, Klatt and Barnes	November 29th 2010	June 2nd 2011

Table 3: Wheel implementation patent. Source: Own elaboration (2024).

Although it was published in 2011, which is indeed a long time ago in the research done, it was found that this design is the only one that with modification could be compatible with the needs of this project, since it allows the incorporation of a steering actuator, an emergency brake, the navigation of normal urban terrains and of course only one wheel.

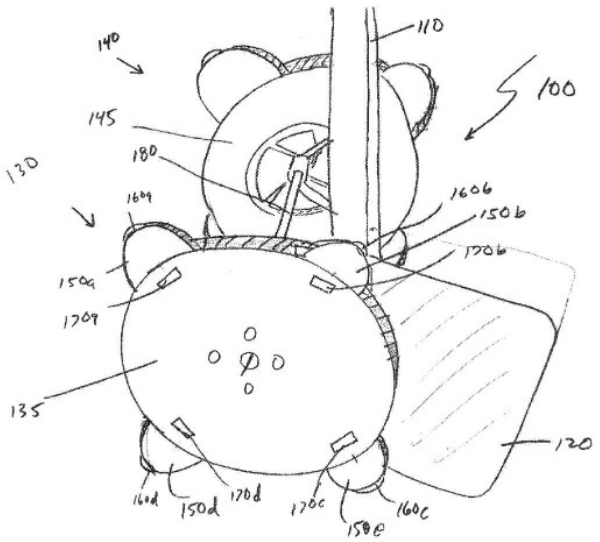


Figure 23: Drawing of a wheel with protrusions. Source: Patent 20110127732 (2011).

Before making the changes needed, a brief explanation must be done. The original design is made up of two seemingly normal wheels that are connected via a metal axle. These two wheels each have small protrusions that can be deployed when stairs are detected using an actuator. When there is no longer a need for them, meaning there are no more stairs the protrusions can be detracted going back to a normal rolling wheel. Now the design must be altered to meet the requirements of this ETA. The first thing that must be changed is the two wheeled design to a one-wheel design. This in theory is very easy, we just have to eliminate one of the wheels but when this is done another problem arises, stability. We must make sure that the one wheel remains stable not only when navigating stairs but when navigating any type of terrain. The first step to achieve this is determining the dimensions of the wheel and protrusions. Before beginning we must now the dimensions of stairs, which will also be important in detecting them. The dimensions used are the average measurements for outdoor stairs in Spain.

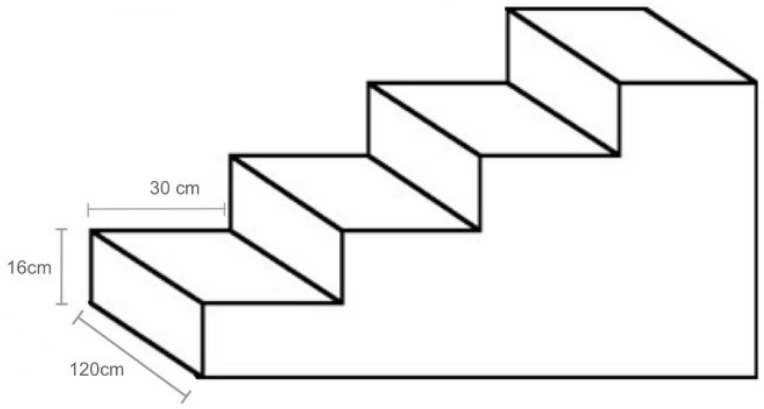


Figure 24: Standard staircase dimensions. Source: Own elaboration (2024).

Therefore, our calculations will be done for a standard staircase with a stair width of 120 cm, a riser height of 16 cm and a tread depth of 30 cm [28]. To determine the wheel diameter, as well as the protrusion dimensions, the equations formulated in the patent will be used, since they can be applied to a single wheel, these are:

$$Depth = R_{wheel} + \left(1 + \frac{\pi}{2}\right) * R_{protrusion} - 1 \qquad (14)$$

$$Rise = R_{wheel} + R_{protrusion} - 1 \qquad (15)$$

The one represents the part of the protrusion that remains housed in the wheel when it is deployed. Applying this to the standard dimensions we arrive at:

$$R_{wheel} = 16.9076\ cm$$

$$R_{protrusion} = 3.6924\ cm$$

To have easier measurements we will round out the values to be 16.9 cm for the wheel radius and 3.7 cm for the protrusion radius. This will leave us with a wheel diameter of 33.8 cm, which is the most important measurement to for the design of the metal bar.

As for the wheel's width we have to make sure that it is wide enough to support the device's weight, as well as remain stable and be able to navigate in all scenarios. The wider the wheel the more stable and more weight it can support but the worse it can maneuver; therefore, it is essential to find the optimal balance. The issue with this, is that the weight depends on the sensors and components used (the exact models), the pressure the user puts on the robot and the exact type of wheel. This is data that is not known and therefore an estimation must be made, which can later be changed if the requirement or are not met during the simulation phase of the project. A good starting point for the width is 10 cm, this dimension will allow the device to be stable and maneuver easily.

3.2.2 BAR

The bar is very simple, and the aim is for it to simulate the white cane shaft but with the exception of the handle. The handle will be designed like the ones used for guide dogs which have a u-shape design. This decision was made due to the fact that these handles allow the user to feel the robot's movement better, which will improve safety and reliability. For the bar there is no need to implement complicated systems like with the wheels, therefore the only necessary thing to do is figure out the dimensions.

[28] Código Técnico (retreived in 2024).

The length of the bar depends mainly on three things: The height of the user, as well as their proportions and the angle at which the bar connects to the wheel/floor. An average white cane should make approximately an angle of 50-degrees with the floor, this is the requirement that will be used for our ETA. As for the length a white cane usually has the same height as the person's sternum but in our case such length is not needed since the device doesn't have to be far away from the user because it has sensors that detect obstacles without having contact with them. For our ETA it is more appropriate to follow the height requirements of a guide dog harness, since the device works more similarly to them. Guide dog owners usually hold the harness at approximately 55% of their height, which is between their waist and lower chest. For our case we will do the calculations for a Spanish adult's average height, which is 169 cm, including both men and women.

$$Height\ from\ the\ ground = 169 * 55\% = 92.95\ cm$$

$$Lenght\ of\ the\ bar\ and\ wheel = \frac{92.95}{\sin(50)} = 121.44\ cm \approx 121.4\ cm$$

$$Lenght\ of\ the\ bar = 121.4 - 33.8 = 87.6\ cm$$

Once this has been calculated we are only left with the dimensions of the handle which was not included as part of the bar length. An average person has a hand width of around 7 to 9 cm, therefore, to be comfortable for all or at least most the width has been set at 12 cm. As for the height it was estimated that 4 cm would suffice, being enough to fit a hand comfortably but not too much that the user wouldn't feel the device's movement.

Taking into account the dimensions of the wheel (without protrusions), the bar and the handle, we are left with the following design.

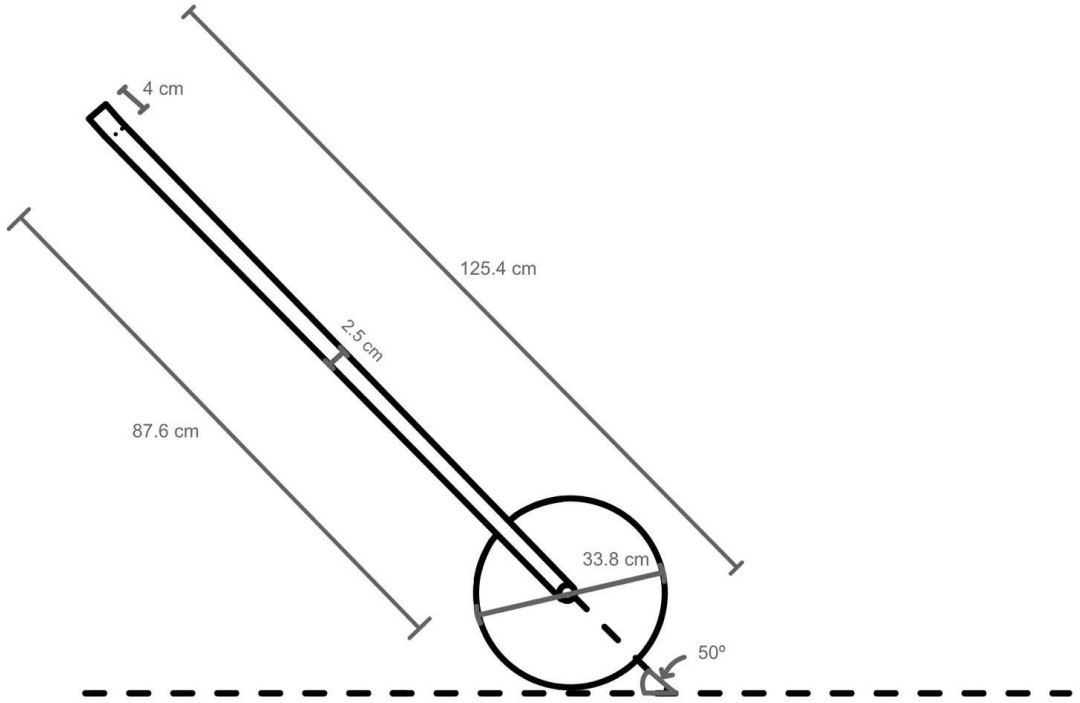


Figure 25: Side view of the ETA. Source: Own elaboration (2024).

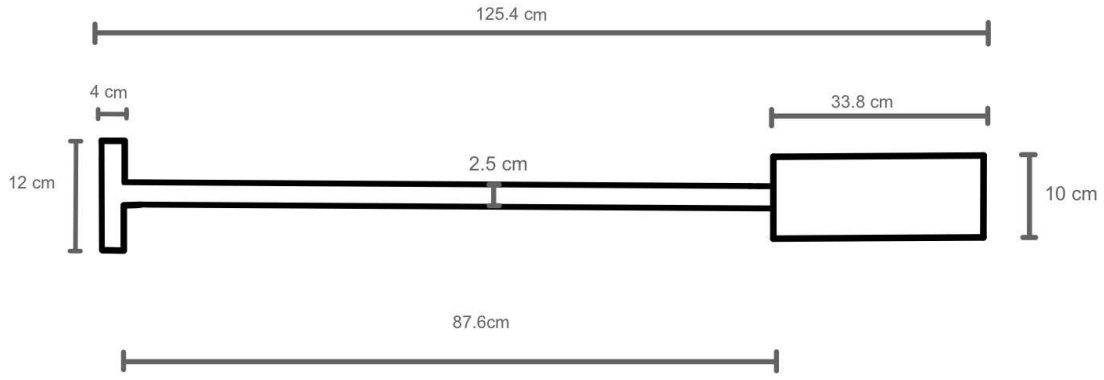


Figure 26: Top view of the ETA. Source: Own elaboration (2024).

3.3 DEVELOPMENT OF TECHNOLOGICAL COMPONENTS

In this section, the sensors and the other technical components will be placed in the device. The aim is to figure out how many sensors are needed and where they must be positioned in the device so that they may achieve their objectives. The sensor selection has been based on the state of the art and the technical components that were analyzed in depth. With that analysis the following sensors have been chosen to be a part of the electronic travel aid:

1.- Ultrasonic sensors: These sensors were chosen instead of the infrared sensors because although they are more expensive, they are more accurate and reliable. Since user safety is the number one priority the extra cost is worth it. They have different purposes and more than one will have to be implemented. The first purpose is to detect obstacles from the user waist level all the way up to their head, such as low hanging tree branches. It will also be used to detect obstacle to the side of the user, such as narrow doorways. Furthermore, they will identify up and down stairs and will be a failsafe in case the other sensors don't detect obstacles.

2.- 2D LiDAR sensor: The LiDAR will be implemented to detect obstacles, mainly ahead of the user. It will map the whole environment, which will allow the device to be extremely accurate and safe.

3.- RGB camera-based sensor: Its purpose is to analyze traffic signs, such as traffic lights or stops. This will allow the user to navigate urban environments independently.

4.- GPS: A global positioning device will not only let the device know where exactly the user is, but it will help guide them to their desired location.

5.- IMU sensor: The device will use this to know the user's orientation so that it may guide them adequately.

6.- Touch sensor: This sensor will be positioned in the handle so that when it does not detect the pressure from the user's hand it can trigger an emergency brake and stop.

Besides these sensors, other technological components must be implemented onto the device to ensure the best guidance and user safety. These are:

1.- Motor: A small motor will be added to help guide the user, as well as provide more torque to go upstairs or navigate difficult terrains. Due to battery consumption and the difficulty of going at the same speed as the user, it will not be constantly on, but only for certain processes.

2.- Single wheel steering actuator: The actuator's function wheel be to steer the wheel in the right direction to guide the user.

3.- Brakes: Brakes are necessary in case of an emergency as another way to let the user know that it is not safe to continue and must stop.

4.- Microphone and speaker: Through the microphone and a voice recognition system the user will be able to communicate with the device and with the speaker the device will be able to communicate with the user.

5.- Vibration motor: This is an extra type of feedback for the user, which will increment safety and communication between the ETA and the individual.

With the components and their functions being specified it is now necessary to actually place them on the physical device shown in figure X. We will start off by placing the sensors which have an added difficulty due to the fact that their field of view and range must be taken into account.

- Ultrasonic sensors: Has an average conic field of view of 30 degrees and a range from a few cm to several meters. The range will not be an issue in this case, but the field of view will be due to the fact that we must use more than one. Using several ultrasonic sensors near each other can cause their signals to interfere with each other giving incorrect values.

- LiDAR: A 2D LiDAR has a range of up to 30 meters (minimum), a horizontal field of view of 360º and a vertical field of view of only a couple centimeters. Therefore, the range will not be a problem and as for the vertical field of view that is why the ultrasonic sensors will be placed to check for objects that are higher or lower than the LiDAR mounting point. Although it will only scan on a set horizontal plain it will be sufficient since most objects will be detected, such as walls, pedestrians and other larger objects.

- RGB Camera: These types of cameras usually have a range of up to 100 meters, which means that range will not be an issue. The horizontal field of view ranges from 60º to 120º, for the purpose of this project. The vertical field of view depends on the horizontal one, as well as the lens of the camera. The camera must be positioned adequately but the ranges are not worrying since its function will be to analyze traffic signs in front of the user.

3.3.1 STAIRS

First and foremost, we will focus on up and down stairs detection, since it is one of the more complex aspects of this project. Although it was stated before a way to potentially go upstairs (without the user having to lift it), since this project is not based on the mechanical side of things the protrusions will not be taken into account. To keep the design as simple as possible for this task several ultrasonic sensors will be used. Several patents and academic papers were studied, and the method chosen was the following [29].

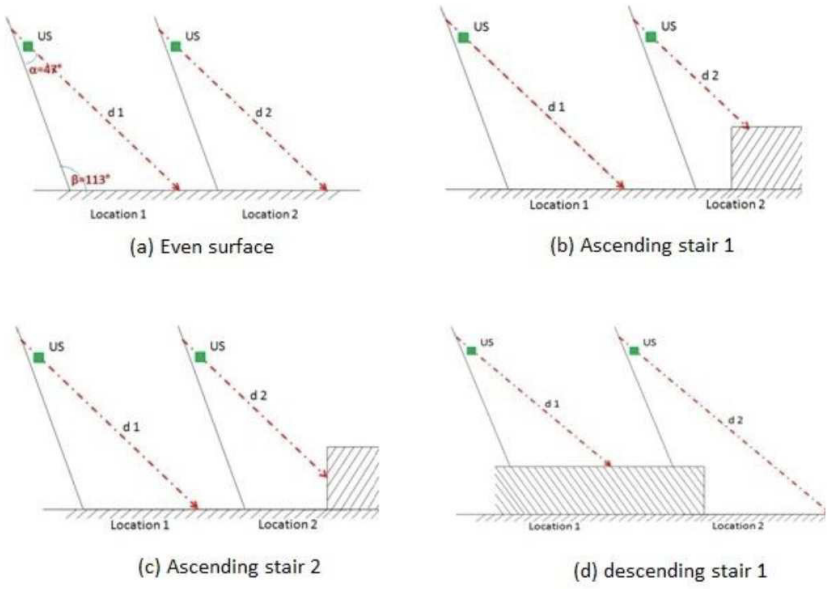


Figure 27: Method for stairs detection.

As can be seen from the image, with this method two distances called d1 and d2 are compared. If the difference between them is what one would expect the rise of a step to be then it can be affirmed that a staircase has been detected. Depending on whether this difference is positive or negative it can be said that the stair is an upward one or a downward one. Although the idea is intuitive and relatively easy to implement it does come with its issues when it's transferred to our own device. Due to the different problems encountered a protrusion has been added to the device and four ultrasonic sensors have been used, called S1, S2 and S3. The final placement is shown in the following image, which will then be explained.

[29] Bouhamed et al (2013).

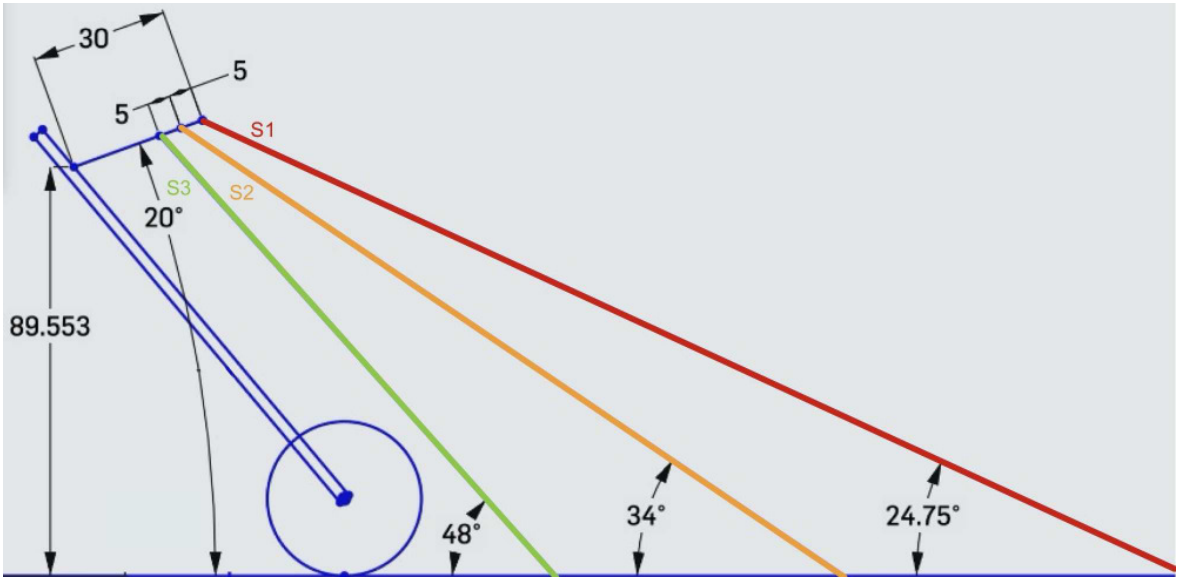


Figure 28: Positioning of ultrasonic sensors, S1, S2 and S3. Source: Own elaboration (2024).

An important clarification to be made is that the waves emitted by the ultrasonic sensors have been represented by a single line instead of a triangle which is what it would look like in reality since these sensors usually have a 30-degree conic field of view. The decision to represent them differently comes due to the fact that they only return the closest distance to them, meaning that the only part that concerns us is the closest line to the stairs.

3.3.1.1 Downward Stairs Sensors

Before arriving at this setup several problems were encountered that had to be solved. The first issue came with identifying downward stairs. In order to detect them an ultrasonic sensor had to be placed at an extremely low angle. This caused two problems, the first being that the distance at which the user could be informed of the stair was very short and two, if the angle was too small the wheel would get in the way of the wave emitted. To overcome this, the physical design had to be slightly modified by adding a protrusion to the bar where the sensor could be better placed. This allows the range of detection to be much larger making the device safer.

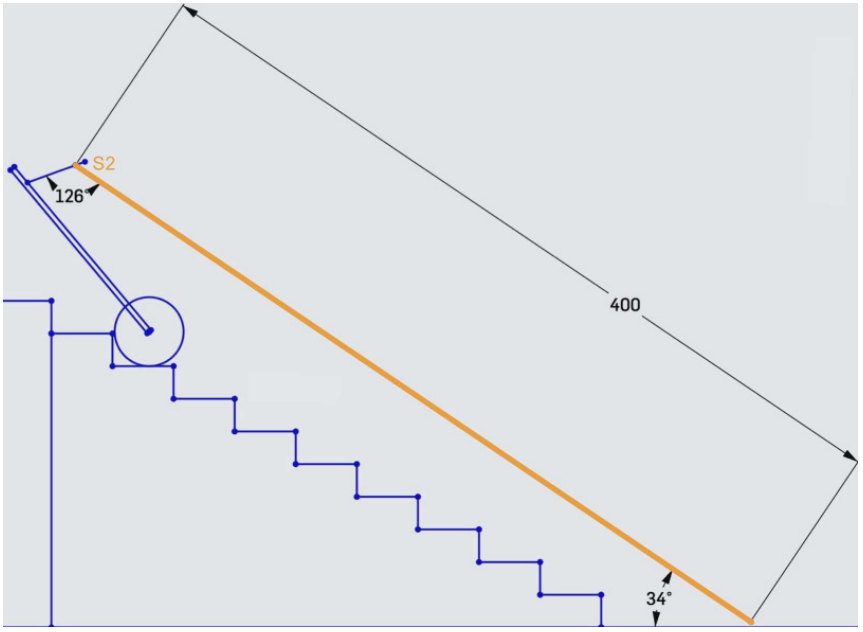


Figure 29: Representation of S2 with downward stairs. Source: Own elaboration (2024).

The problem with placing S2 at a 34º angle was that once the device was lowered onto the downward staircase the sensor would lose range (didn't detect anything). Due to this a third sensor, S3 was added. This sensor has a larger angle with the floor and a smaller one with the bar which allows it to never loose range. It always detects either the floor or one of the stair steps.

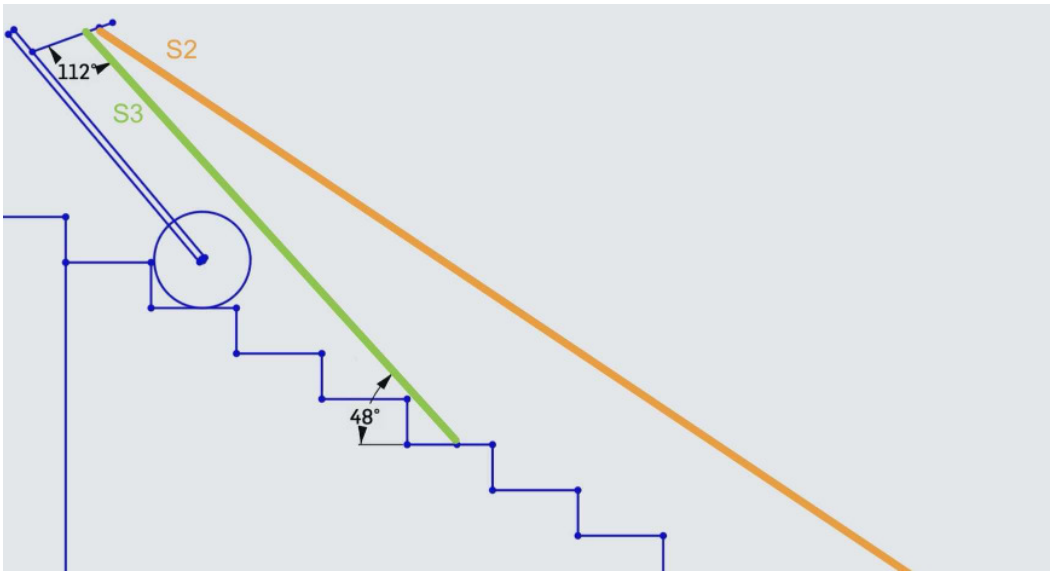


Figure 30: Representation of S2 and S3 with downward stairs. Source: Own elaboration (2024).

This configuration allows the device to detect the first downward step from a distance of 109.5 cm which allows the user sufficient time to receive the information and act accordingly.

3.3.1.2 Upward Stairs Sensors

Now to detect upward stairs another ultrasonic sensor was added, because although for downward stairs a very steep angle is needed, for these it is not. Not only is it not needed but it is not optimal, since it would make the range of detection a lot shorter than it could be. For this reason, S1 was placed on the protrusion in such a way that it makes a smaller angle with the floor and a larger one with the bar. This allows the sensor to identify upward stairs from further away. Due to this modified angle S1 will never fall out of range so there is no need to add another detection sensor.

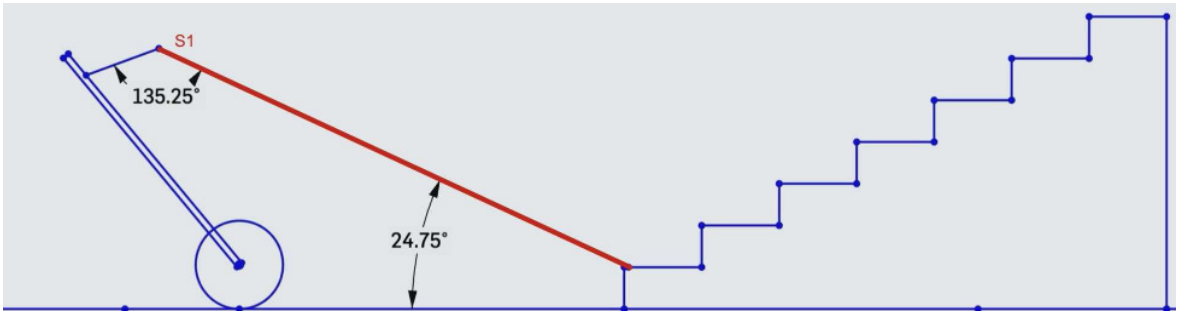


Figure 31: Representation of S1 with upward stairs. Source: Own elaboration (2024).

This configuration allows the device to detect the first upward step from a distance of 150 cm which allows the user sufficient time to receive the information and act accordingly.

3.3.1.3 Algorithm for Stair Detection

With the sensors positioned correctly the only thing left to do is to actually elaborate the mathematics behind the stair detection. For this we first need to assign names to the distances and angles.

$$distance\ measured = d_i$$

$$horizontal\ distance = d_{i,h}$$

$$vertical\ distance = d_{i,v}$$

$$vertical\ distance\ to\ the\ wheel/ground = d_{Si}$$

$$horizontal\ distance\ to\ the\ wheel = d_{Si,w}$$

$$distance\ from\ where\ the\ sensor\ hits\ the\ step\ to\ the\ start = d_{i,s}$$

$$horizontal\ angle = \theta_{i,h},$$

$$vertical\ angle = \theta_{i,v}$$

$$for\ i = 1,2,3$$

For a more comprehensive understanding the following images were created.

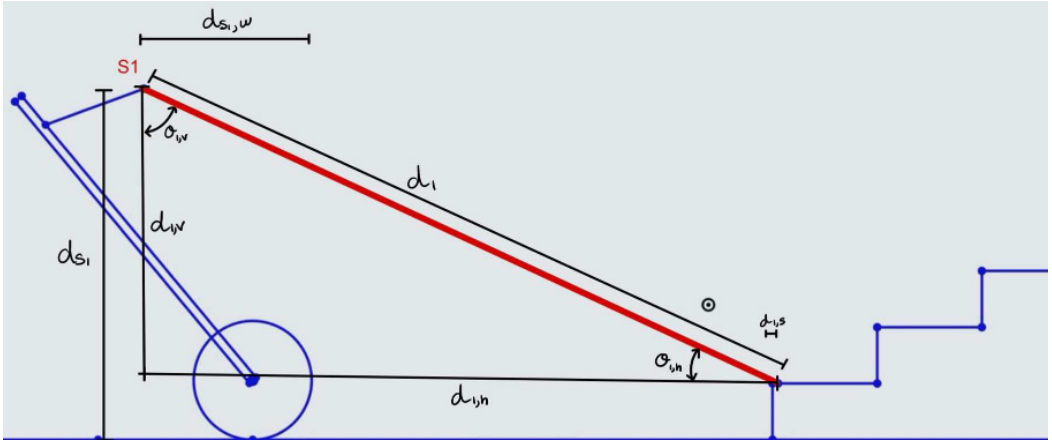


Figure 32: Names assigned to angles and distances of S1. Source: Own elaboration (2024).

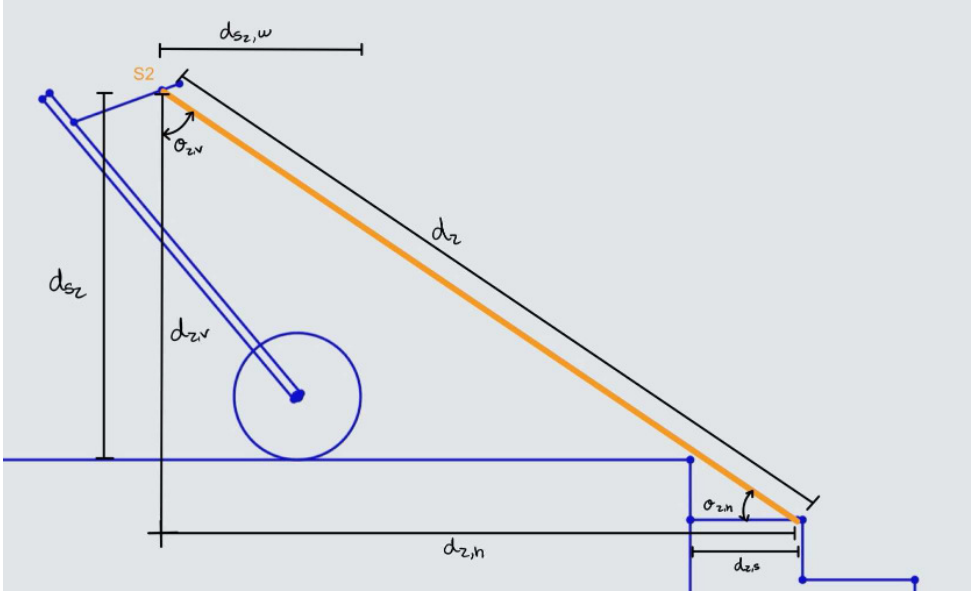


Figure 33: Names assigned to angles and distances of S2. Source: Own elaboration (2024).

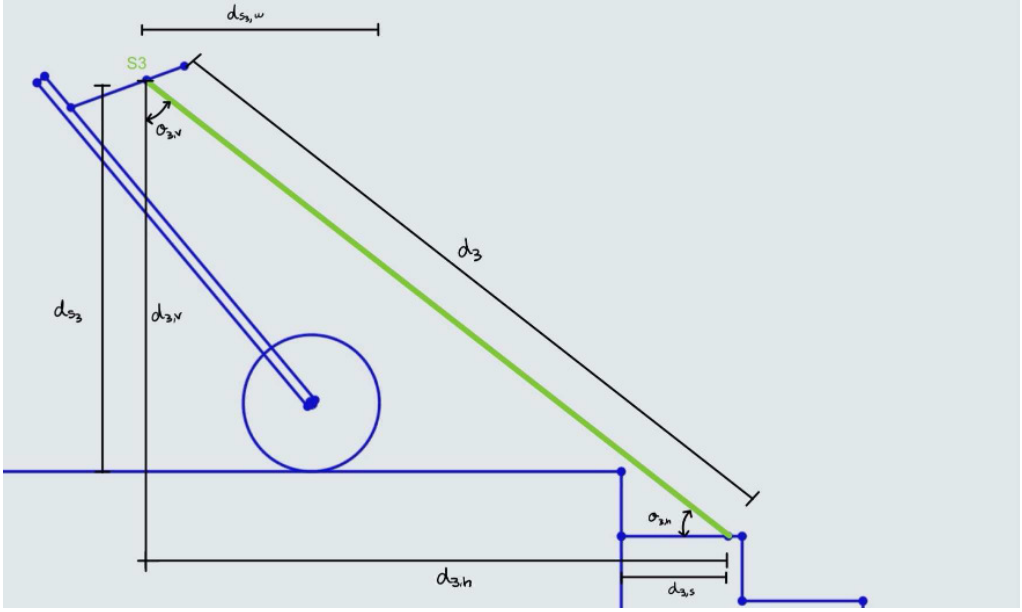


Figure 34: Names assigned to angles and distances of S3. Source: Own elaboration (2024).

With the names assigned we can continue. The first equation needed is that of the ultrasonic sensor which was mentioned in a previous chapter.

$$d_i = \frac{time\ of\ flight * speed\ of\ sound}{2} \tag{16}$$

The equation for the speed of sound is:

$$speed\ of\ sound = \sqrt{\frac{\gamma * R * T}{M}} \tag{17}$$

Since we assume that the waves emitted by the device will always travel through gas (air) the only variable that is not constant is the temperature. Therefore, to make the ETA even more accurate a thermostat can be added in order to use the correct temperature value in the previous equation.

Once the value of d_i is determined, the horizontal and vertical distances can be easily calculated using basic trigonometry since the angles are known.

$$d_{i,v} = d_i * \sin\theta_{i,h} \tag{18}$$

$$d_{i,h} = d_i * \cos\theta_{i,h} \tag{19}$$

$$for\ i = 1,2,3$$

As previously stated in order to detect stairs two distances must be compared, which are $d_{i,v}$ and d_{Si}. By subtracting one from the other the difference between the two is calculated which is proportional to the difference between $d_{i,v}$ and the ground. Therefore, if that distance is proportional to the expected measurement of a standard rise in a staircase, then it can be affirmed that one has been detected.

$$if \begin{cases} \frac{d_{Si} - d_{i,v}}{N} = 16 \pm 2, & upward\ stairs\ detected \\ \frac{d_{Si} - d_{i,v}}{N} \neq 16 \pm 2, & no\ stairs\ detected \end{cases} \quad for\ i = 1\ ;\ N = 1, \ldots.,15 \tag{20}$$

$$if \begin{cases} \frac{d_{Si} - d_{i,v}}{N} = -16 \pm 2, & downward\ stairs\ detected \\ \frac{d_{Si} - d_{i,v}}{N} \neq -16 \pm 2, & no\ stairs\ detected \end{cases} \quad for\ i = 2,3\ ;\ N = 1, \ldots.,15 \tag{21}$$

To have more accurate detection rates an error of ± 2 has been added in case a set of stairs has a larger or smaller rise than the standard. As for the value of N, it was estimated that due to the range and angle constraints no more than 15 steps could be reached, which means that it would not be possible for the difference in distance to be more than 15 times the rise of a single step. It must also be noted that S2 can lose range when the device is lowered onto the stairs, so to avoid false detections the previous calculations will only be done if $d_{s,v} \neq 0$.

Once the staircase is identified either downward or upward, the horizontal distance can be used to tell the user how far away it is. The one problem with this is that since there is a delay between the start of the step and the actual moment of detection, the horizontal distance does not point to the start but further away. This difference is especially noticeable when detecting downward staircases which is why its equation must be different. The solution to this is quite simple, subtract the distance from the beginning of the step to where the sensor actually hits it from the horizontal distance. To have an additional security method another 5 cm will be added so that the blind individual never thinks the first step is further away than it truly is which could cause serious injuries. This way when they have a small range at which the stairs could start and can use the actual device to know the exact location, by using it much like they would a white cane but without the oscillating motion.

$$distance\ to\ the\ start\ of\ the\ staircase = d_{i,h} - d_{Si,w} - d_{i,s} - thread * N - 5 \qquad (22)$$

$$for\ i = 1, 2, 3 \quad ;\ for\ N = 1, 2, \ldots, 15$$

The last concern is how to inform the user how far away they are from the end of the stairs. An easy way to know if the sensor has reached the end is by seeing if the difference between d_{Si} and $d_{i,v}$ is getting smaller, since this difference is zero when travelling on flat ground. This can be done by using the previous calculation where the number N that solves the equality is saved. That N represents the number of steps between the user and the end of the sensor, if this number gets smaller it must mean that the user is getting closer to the end because the vertical distance is decreasing. Therefore, the device can alert the visually impaired individual that there are X number of steps left till they reach flat ground.

3.3.1.4 Ultrasonic Sensor Interference

There is still one very important issue that has not been addressed, which is the interference caused by the fact that more than one ultrasonic sensor is being used. As stated before, although in the drawings the waves emitted were represented as a line in reality they are emitted in a conic shape and since they are in such close proximity almost all of their field of view overlaps with one another. This causes a problem because all of them emit the same type of waves, ultrasonic and therefore the sensors can confuse another one's signal as their own, causing false readings. There are ways to solve this issue, the easiest one is to place the sensors at a distance and angle specific so that their

fields of view don't overlap, but as seen previously stair steps are too small for this to be possible. Another option is applying some type of algorithm that would fix the problem, out of the ones analyzed it was found that the best option is to use multiplex operation. This allows the sensors to emit waves at different times in a sequential order which avoids any overlapping. However, this does come with a disadvantage, the reaction time. Since only one sensor can work at a time and in our case each sensor has a different function the time it takes for them to successfully perform their assigned task increments [30]. For example, it S1 is active that means that S2 is not, and if in that precise moment a downward staircase entered the detection range it would not be identified until S2 was activated again. Although this could be an issue, since we are only using three sensors the emitted waves will alternate at a high enough speed that the time lost won't be dangerous for the user.

The following equations will develop the mathematical model behind the idea of using multiplex operation, which in simpler words means doing a sequential timing of the ultrasonic sensors signals.

$$Total\ cycle\ time\ =\ T$$

$$Time\ given\ to\ each\ sensor\ =\ T_i \quad for\ i\ =\ 1,2,3$$

$$T\ =\ \sum_{i=1}^{3} T_i \qquad (23)$$

The time given to each sensor will be equal for all and to make sure all interference is avoided its value will be the maximum amount of time it could take for a wave to be emitted, reflected from an object and received by the sensor. To calculate this the basic ultrasonic equation will be used with a distance of 4 m which is the maximum range and therefore will give us the maximum time. But a small change has to be made, which is instead of diving by two we have to divide by four since we are looking for the time of the round trip.

$$d_{max}\ =\ \frac{Ti\ *\ speed\ of\ sound}{4}\ \rightarrow\ T_i\ =\ \frac{d_{max}\ *\ 4}{speed\ of\ sound}$$

The result from this equation which will depend on the speed of sound will be the time allotted to each sensor. The sequence will be circular, starting with S1 to S2 to S3 and then starting again.

[30] Pepperl-Fuchs (2019).

3.3.2 GENERAL OBSTACLE DETECTION

The main component used for general obstacle detection will be the 2D LiDAR sensor. As explained at the beginning of this section this type of LiDAR has a horizontal field of view of 360-degrees but only a few degrees for the vertical one. This means that the placement must be seriously though about, in order to detect as many obstacles as possible. To place it correctly it has to be taken into account the environment in which our ETA will be used in, which is an outdoor urban one. Furthermore, the LiDAR will not be used to detect ground level or head level obstacles, since we already have three ultrasonic sensors for ground level and a fourth one will be added pointing up for head level. Therefore, the LiDAR has to detect obstacles at a mid-height (waist level) which is around 0.9 to 1.2 meters from the ground. This location will allow the sensor to identify pedestrians (adults and kids), cars, benches, trashes, walls, bollards, bicycles, etc. Almost every item that the individual could encounter.

With this an issue presents itself, and that is that the top of the bar is 96 cm from the ground and the top of that has been occupied with the stair detecting ultrasonic sensors (located at 90 cm). This reduces the range of locations in which the LiDAR could be positioned at, meaning that it will have to be lower than the optimal stated before. This loss of height can be solved by mounting the sensor at a slight upward angle, making sure the added protrusion does not interfere with it. With this we arrive at the following position for the LiDAR assuming a range of 600 cm:

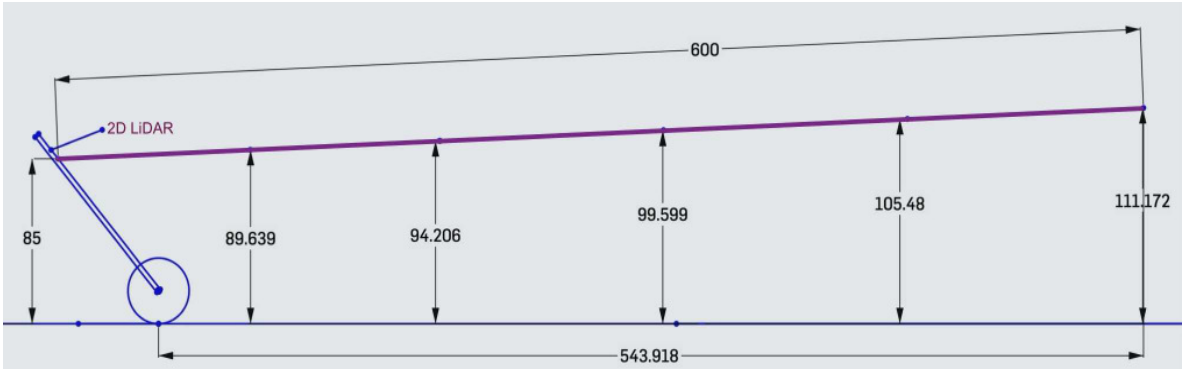


Figure 35: Positioning of the 2D LiDAR. Source: Own elaboration (2024).

With this setup the mounting of the LiDAR makes a 2.5-degree angle with the horizontal allowing it to cover the needed vertical range expressed previously. In addition to this we have the three ultrasonic sensors which have a double function of detecting stairs and other ground level obstacles, meaning that even if they don't detect a staircase, they can still inform the user of an anomaly detected.

Previously, in state of the art it was explained how a LiDAR works but without much detail. Therefore, in the following pages we will dive deeper into the ins and outs of a 2D LiDAR and how to implement it in our ETA. The basic equation for it was identified before and is the following.

$$Distance\ to\ Object = \frac{Total\ Time\ of\ Flight\ \times\ Speed\ of\ Light}{2} \tag{24}$$

The LiDAR emits laser pulses by doing a sweeping motion across a horizontal plane and measures the time it takes for each one of them to come back. This allows the sensor to create a point cloud of the environment, since it knows all of the points and distances in that plain.

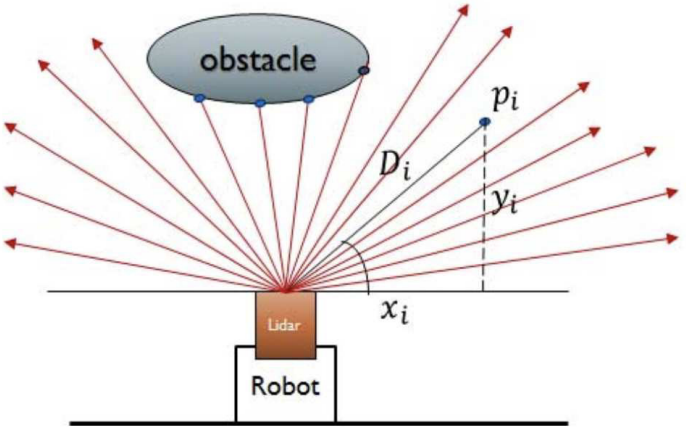


Figure 36: Point cloud of a 2D LiDAR sensor.

As seen in the image, the issue is that the position of each laser point is expressed in polar coordinates, so the first step is to convert them into Cartesian coordinates. For this we must assign names to each variable.

$$distance\ from\ the\ sensor\ to\ the\ laser\ point\ =\ D_i$$

$$position\ of\ the\ laser\ point\ =\ P_i$$

$$angle\ of\ the\ laser\ beam\ with\ the\ horizontal\ =\ \alpha_i$$

$$angle\ of\ the\ laser\ beam\ with\ the\ vertical\ =\ \beta$$

Conversion to Cartesian coordinates.

$$x_i = D_i * \cos\alpha_i * \cos\beta \quad (25)$$

$$y_i = D_i * \sin\alpha_i * \sin\beta \quad (26)$$

$$for\ i \in (-180, 180)$$

$$for\ \beta = 2.5°$$

The vertical angle will be constant since the LiDAR is fixed to the bar and the horizontal angle will vary from -180 to 180 degrees because the sensor has a horizontal field of view of 360-degrees.

Once the distances are obtained an obstacle detection algorithm must be applied, since without it the ETA would not be able to use this information correctly. After analyzing different papers, it was found that the best method to do this is obstacle segmentation and clustering.

Segmentation in this context consists in grouping the data returned by the sensor, this grouping is done by rotation. The LiDAR scans the plain by rotating, and in that rotation x number of beams are emitted. Those beams then return as x distances or the same, x points. With this segmentation those x points are grouped into one, in order to organize the data received. On the other hand, clustering also means grouping points but this time using closeness criteria. If two laser points are near to one another then it can be assumed that they form one single distinct object. So, ¿what is the closeness criteria? There are many different criteria for this, but since the one of the aims of this ETA is to make everything as simple as possible the criteria, we will implement this in a very basic way, dividing the process into two:

1.- Obstacle not detected: If the laser beam emitted does not return to the device this means that no obstacle was encountered since no reflection occurred. When this is the case those imaginary points that appear at the maximum distance, which is the LiDAR's range are eliminated. This way the number of points that have to be analyzed to be clustered by the closeness criteria decreases

2.- Width of the robot criteria: If the distance between two points is less than the width of the robot they are grouped into one object and if the distance is larger, they are considered two different objects. This may seem too broad of a classification, but it perfectly fits our needs. If the ETA and the person do not fit between two objects, then that path is not safe for the user, and it must be avoided in the same way as if it was only one object.

Both of these criteria allow us to simplify the map of the environment and therefore the obstacle avoidance process. [31]

The final piece of the puzzle for obstacle detection is identifying head level objects, for this an extra ultrasonic sensor called S4 will be placed on the protrusion.

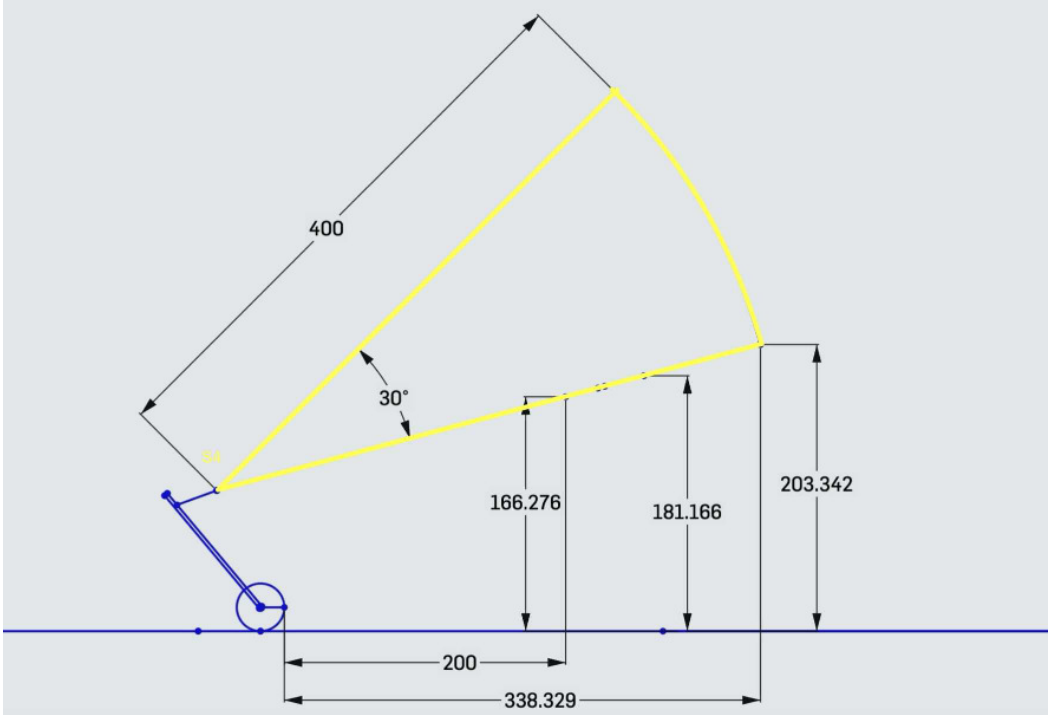


Figure 37: Positioning of ultrasonic sensor S4. Source: Own elaboration (2024).

The sensor is placed on the end of the protrusion at a 15-degree angle with the floor. If we assume that the user has a height of around 169 cm then this setup will allow them to know that they will hit an object 2 meters before they actually do. In addition, with this positioning S4's field of view will not overlap with the fields of view of the rest of the sensors and thus there is no need for sequential timing.

With this final sensor, the ETA is able to detect all obstacles in front of the user from ground level all the way up to 2 meters.

[31] Ghorpade et al (2017).

3.3.3 TRAFFIC LIGHT DETECTION

The final part of this section is traffic light detection which will be done with an RGB camera. As stated previously for this project we will assume the average fields of view which are, 90-degrees for the horizontal and 60-degrees for the vertical. For the range although it could be incredibly long (up to 100 meters) for this project that kind of distance is not needed, and a 25-meter range will be sufficient since the longest crosswalk in Spain is that length. Reducing the range will allow for the device to be cheaper, since it will need less resolution. Before continuing it is necessary to figure out what the horizontal and vertical field of view of the camera will be. Knowing that a 30-meter range device will be used, the average fields of view are 68-degrees for the horizontal and 40-degrees for the vertical.

To actually position it the height of traffic lights and the length of the crossing has to be taken into consideration. In Spain a pedestrian traffic light must have a height between 2 and 2.4 meters, so the device must be able to identify it in these range of heights. As for the length of the crossing since this one varies a lot from one to another it is essential that the device is able to detect all traffic lights of from a distance of 3 meters (minimum length) up to 30 meters. This is not an issue at all since the field of view is extremely wide and gets wider the further away the sign is. The following image shows the camera's position, as well as a representation that the traffic sign will always be in its field of view.

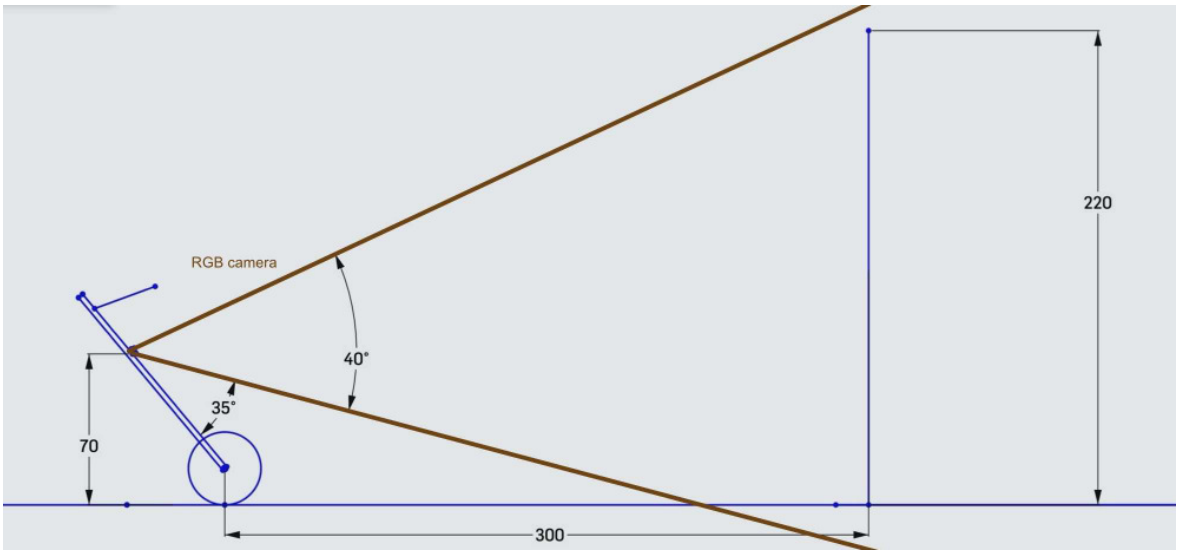


Figure 38: Positioning of RGB camera. Source: Own elaboration (2024).

Since the only function that the camera has is to detect whether or not a traffic light is green or red, how it works is very simple. There are two things that work together in order to make the detection possible, these are ROI and HSV color.

ROI [32] stands for region of interest and the idea behind it is that only a certain part of the image is selected for processing. This not only reduces the computational complexity of the program, but it also reduces the possible errors made, such as confusing surrounding areas by a traffic light. To apply this a very simple process must be followed and the only needed known variables are the width and height of the cameras field of view, which we will call W and H, respectively. Once these are determined, the top-left corner (x_1, y_1) and bottom-right corner (x_2, y_2) of the ROI can be specified, which will allow the computer to build a new image with those boundaries. For the height of the ROI, we know that the traffic light will always be in the top half plain and for the width of the ROI, since the sign can be in the center or to either side, it cannot be reduced. The following equations represent the boundaries of the new field of view, setting the origin at the top left of the image.

$$x_1 = 0 \qquad (27)$$

$$y_2 = 0 \qquad (28)$$

$$x_2 = W \qquad (29)$$

$$y_2 = \frac{H}{2} \qquad (30)$$

HSV [33] is a color space that stands for hue, saturation and value. It is a cylindrical representation of colors that separates the image luminance form the chromaticity, which makes it more intuitive and effective for color detection and segmentation, exactly what this project is trying to achieve. For a more detail explanation, the three components will be analyzed:

- Hue: It represents the type of color by having its value vary from 0 to 360 degrees, where each degree corresponds to a different color. In conclusion, it describes the pure pigment of a color without taking into account lightness or saturation.

- Saturation: It measures the amount of color intensity, so its vibrancy. It ranges from 0% to 100%, with 0 meaning that the color is dull and 100 meaning that it is vivid.

- Value: It represents the lightness or darkness of a color, and ranges from 0%, completely black to 100%, completely white.

[32] MathWorks (retrieved in 2024).
[33] Wonghabut, Pasit, et al (2018).

The first step is to do the conversion from RGB (our camera's color space) to HSV, for this the RGB values must be normalized, by dividing them by their range.

$$R_{nor} = \frac{R}{255} \tag{31}$$

$$G_{nor} = \frac{G}{255} \tag{32}$$

$$B_{nor} = \frac{B}{255} \tag{33}$$

Then the hue (H), saturation (S) and value (V) can be calculated.

$$V = max(R_{nor}, G_{nor}, B_{nor}) \tag{34}$$

$$S = \begin{cases} 0 & if\ V = 0 \\ \dfrac{V - min(R_{nor}, G_{nor}, B_{nor})}{V} & if\ V \neq 0 \end{cases} \tag{35}$$

$$H = \begin{cases} 0 & if\ S = 0 \\ 60° * (\dfrac{G_{nor} - B_{nor}}{V - min(R_{nor}, G_{nor}, B_{nor})}) & if\ V = R_{nor} \\ 60° * (2 + \dfrac{B_{nor} - R_{nor}}{V - min(R_{nor}, G_{nor}, B_{nor})}) & if\ V = G_{nor} \\ 60° * (4 + \dfrac{R_{nor} - G_{nor}}{V - min(R_{nor}, G_{nor}, B_{nor})} & if\ V = B_{nor} \end{cases} \tag{36}$$

Once the conversion has been done, these values can then be compared to the ideal ones for red and green in order to determine the color. The ideal values are represented in the following table. [34]

Traffic Light Color	**Hue (H)**	**Saturation (S)**	**Value (V)**
Red	329.76º - 36º	70% - 100%	55% - 100%
Green	158º - 177.98º	73.6% - 100%	19.6% - 96.5%

Table 4: HSV values for green and red. Source: Hassan, Nazirah; Ming, Kong Wai; Wah, Choo Keng (2020).

[34] Hassan, Nazirah; Ming, Kong Wai; Wah, Choo Keng (2020).

With ROI and HSV the ETA will be capable of detecting the color of a traffic light and therefore act accordingly. Finally, this process will only occur if the path guiding system which will be explained in the next section confirms that there is a crosswalk ahead.

3.3.4 OVERALL SENSOR POSITION

As for the rest of the components since they do not have a range or field of view that must be taken into account their placement does not have to be specified. With this, we arrive at the final design which is shown in the following image.

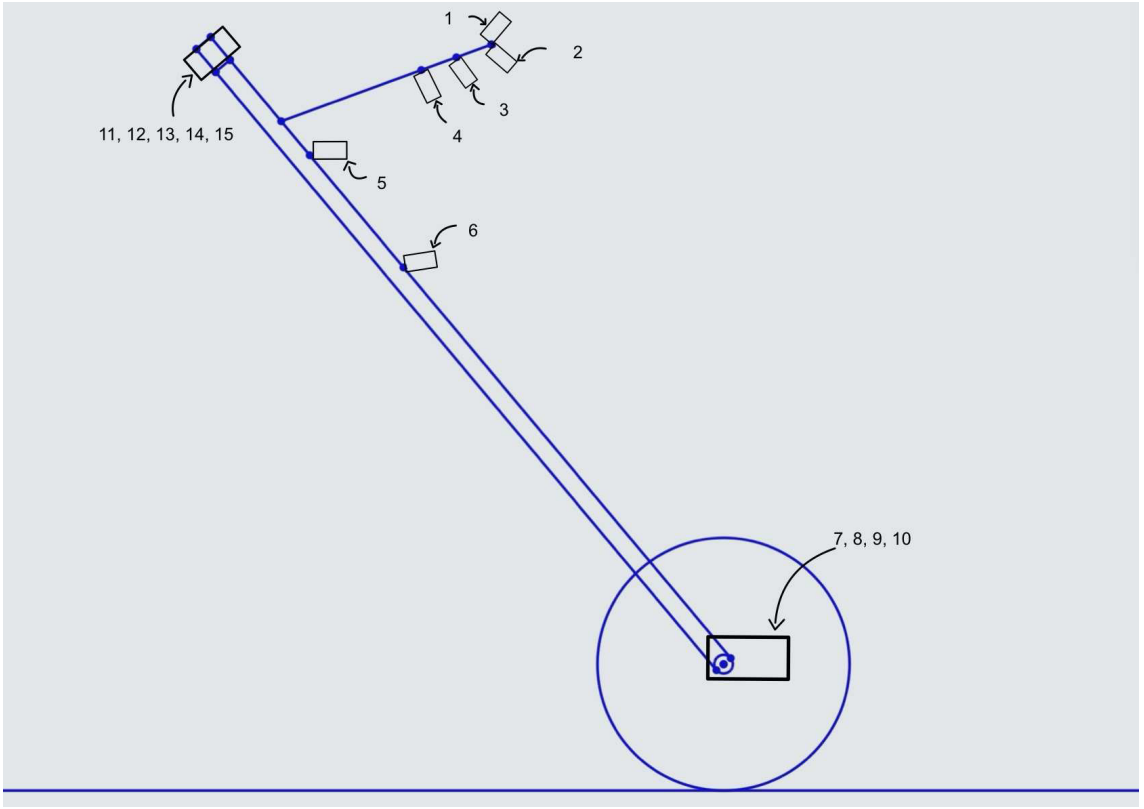


Figure 39: Positioning of all of the technological components. Source: Own elaboration (2024).

Within this image numbers 1, 2, 3, and 4 refer to the ultrasonic sensors, 5 represents the 2D LiDAR and 6 is the RGB camera. As for the blocks that include more than one number, we have 7, 8, 9, and 10 which represent the motor, brake, steering actuator, and IMU sensor. Finally, 11, 12, 13, 14, and 15 refer to the microphone, speaker, touch sensor, vibration motor, and GPS system.

Once the positions and final components are known the circuitry can be elaborated, due to the number of sensors needed the best option is to use a Raspberry Pi 4 [35]. This is a portable, affordable, and powerful single-board computer with very high processing power. The most interesting ports and connectivity characteristics that this Raspberry has for this project are the following:

- Has forty pins, with twenty-six of them being GPIO (General Purpose Input/Output) and has both 3.3 V and 5 V power pins. This means that it has enough space to connect all of the different sensors needed.

- One CSI Port (Camera Serial Interface) which is used to connect camera modules, something essential for this project for traffic light detection.

- Audio/Video Output, this allows us to connect an external audio source with the necessary voice commands to provide feedback to the user.

- It also includes a wireless network and Bluetooth, which increases the connectivity options

- It can have up to 8GB of RAM which would be sufficient to store any maps needed.

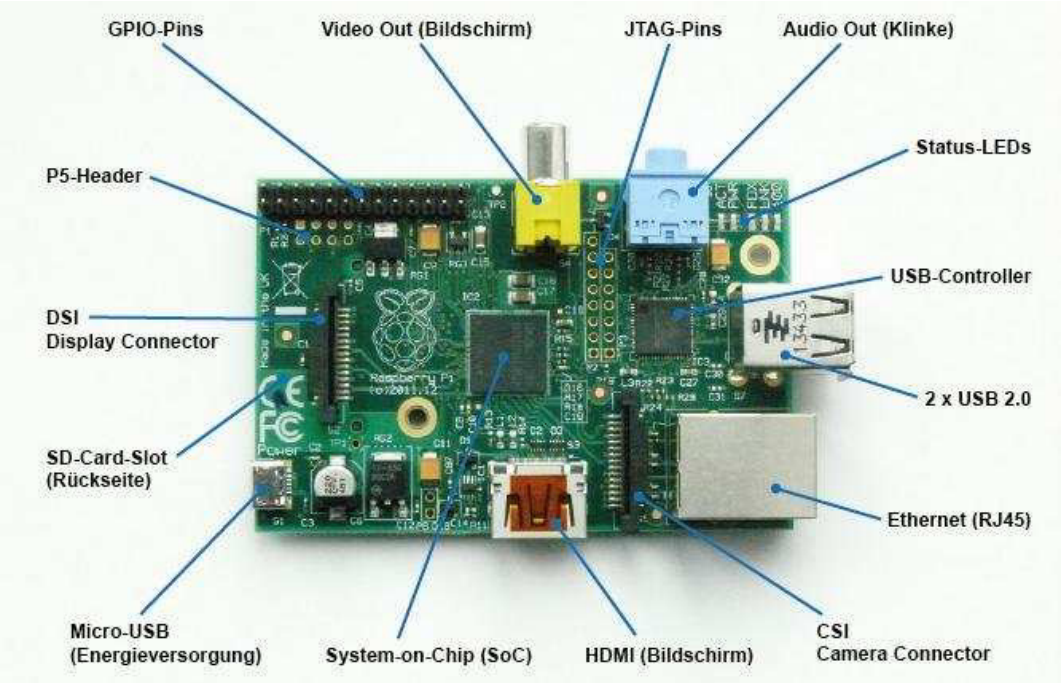


Figure 40: Raspberry Pi 4 image.

[35] Nayyar, Anand; Puri, Vikram (2015).

As seen on Figure X, this single-board computer has a lot more applications that were not specifically mentioned in the previous text. Although not needed as of now if this project were to keep progressing and other functions were to be implemented the Raspberry Pi 4 would be capable of handling it. To elaborate the circuitry, it is important to know the characteristics of each of the sensors and modules that are used, to connect them to the correct port. For this we need to know the exact sensor model that will be used, which will be the ones stated in Chapter 2: State of the Art. As for the other technological components, since they do not need a certain range or field of view they can be chosen more generically.

1.- Raspberry Pi Camera Module V2 [36]:

- Can be connected directly to the CSI Port.

2.- RPLIDAR A1 [37]:

- Needs a 5 V power source, so it can be connected to a VCC pin.
- Needs ground, it can be connected to a GND pin.
- TX/RX serial communication, this pin connection is so that the LiDAR is able to transmit data to other devices, as well as receive it. For this any GPIO pin can be used.

3.- HC-SR04 [38]:

- Needs a 5 V power source, so it can be connected to a VCC pin.
- Needs ground, it can be connected to a GND pin.
- TRIG, this is to trigger or send the ultrasonic wave and can be connected to any GPIO pin.
- ECHO, this is to receive the reflected ultrasonic wave and can be connected to any GPIO pin.

4.- Motor and Motor Driver [39]:

- The motor will have to be connected to an external power source such as a battery, which will depend on the motor specifications.
- Needs ground, it can be connected to a GND pin.
- IN1/IN2, these are the motor driver's control pins and can be connected to any GPIO pin.
- OUT1/OUT2, these are the motor driver's outputs which go to the motor terminals.
- EN, this is an enable pin which allows to control the motor's speed and can be connected to the board using any of the GPIO pins.

[36] Raspberry Pi (retrieved in 2024).
[37] Slamtec (retrieved in 2024).
[38] Cytron Technologies (2023)
[39] SparkFun (2023)

5.- Speaker and USB Sound Card [40]:

- The sound card with the necessary voice command to provide feedback can be connected to a USB port.
- This can be connected to the audio output of the USB sound card.

6.- Microphone [41]:

- It can be connected to the audio input of the USB sound card.

7.- Vibration Motor [42]:

- Needs a 5 V power source, so it can be connected to a VCC pin.
- Needs ground, it can be connected to a GND pin.
- A control pin must also be added to regulate the motor, and this can be connected to any GPIO pin.

8.- GPS Module [43]:

- Needs a 5 V or 3.3 V power source depending on the model, so it can be connected to any VCC pin.
- Needs ground, it can be connected to a GND pin.
- TX/RX serial communication, this pin connection is so that the GPS is able to transmit its data to other devices, as well as receive it if need be. For this any GPIO pin can be used

9.- Inertial Measurement Unit [44]:

- Needs 3.3 V power source, so it can be connected to a VCC pin.
- Needs ground, it can be connected to a GND pin.
- The IMU needs another type of communication called Inter-Integrated Circuit or I2C for short which is used to connect various sensors with minimal wiring. The main reason why the IMU need it is because it is made up of three other sensors and this way they can share the communication channel, which simplifies the circuitry. This is done by connection the device to an SDA pin (data) and an SCL pin (clock), this last one is to synchronize the signal between the different sensors. In addition, two resistors are needed, usually of 4.7 kΩ to avoid having an exceedingly high-power consumption.

[40] Raspberry Pi (retrieved in 2024).
[41] Raspberry Pi (retrieved in 2024).
[42] Pololu (retrieved in 2024).
[43] Adafruit (retrieved in 2024).
[44] Adafruit (retrieved in 2024).

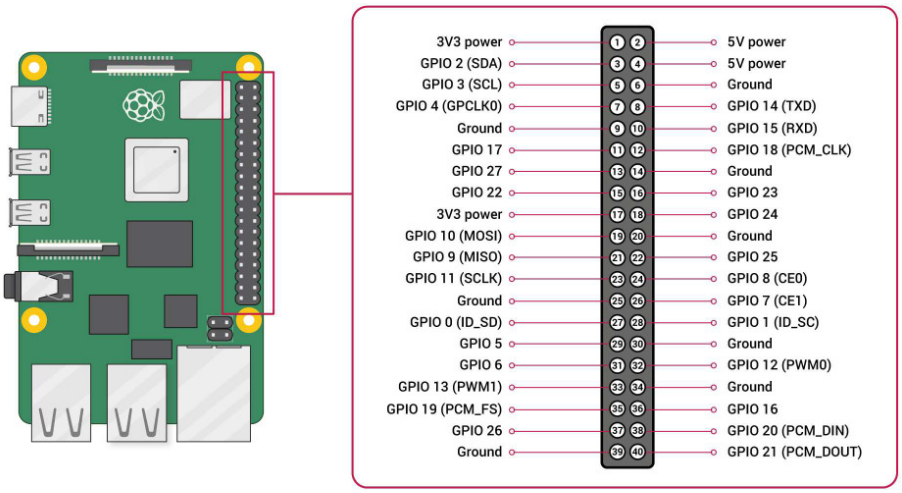


Figure 41: Raspberry Pi 4 pin location. Source: Adafruit (retrieved in 2024)

With all of this information in mind and using the pin locations shown in Figure 37 locations the circuitry map can be developed.

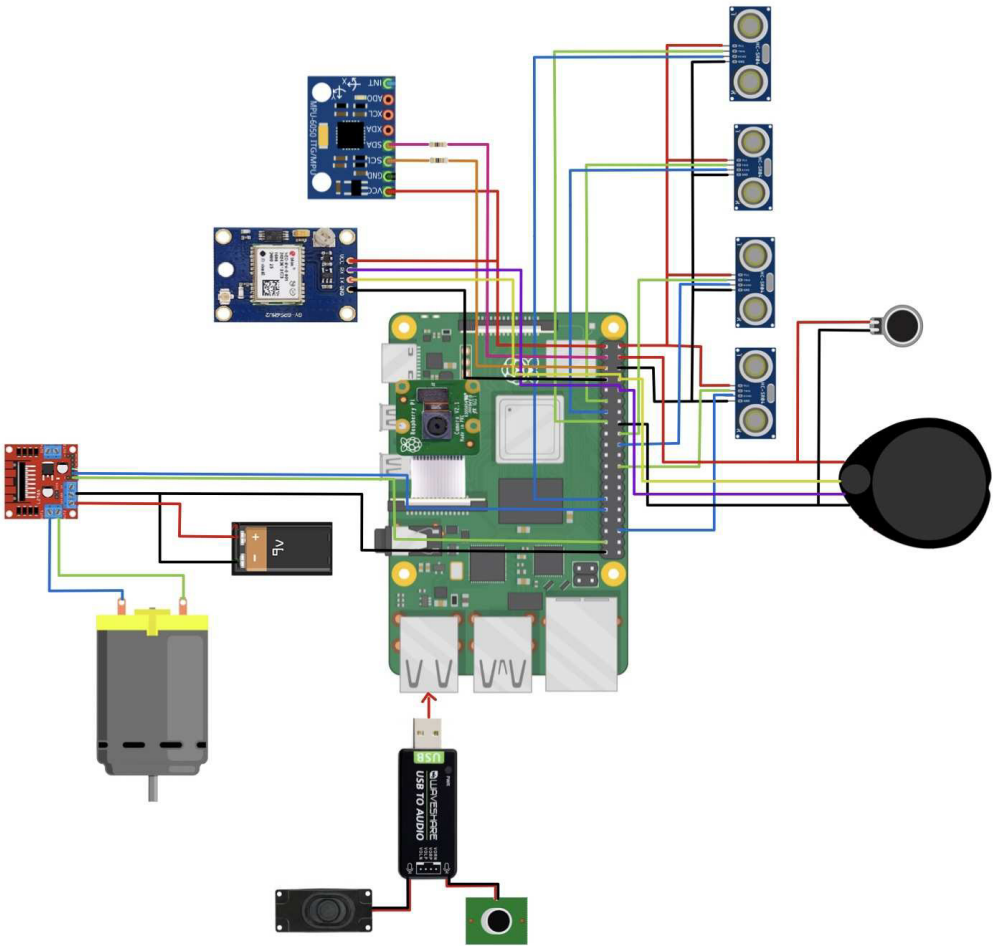


Figure 42: Circuitry map. Source: Own elaboration (2024).

With this map, the circuit can easily be implemented into the ETA by extending the cables so that each of the components can be placed correctly.

3.4 PATH GUIDING AND OBSTACLE AVOIDANCE

Now that the sensors and the obstacle detection process has been explained, it is time to dive into the path guiding and obstacle avoidance process. The most important objective of this ETA is to guide the visually impaired individual safely from point A to point B in an urban environment, in order to achieve this a path guiding and obstacle avoidance algorithm must be implemented. This section will be divided into two, a part to determine the global path guiding method and another one for the local path guiding. The global method will allow the robot to determine the shortest overall path to the desired destination without considering possible obstacles in the way. The local method will override the previous one when the ETA detects an obstacle in the path to ensure it avoids it.

3.4.1 GLOBAL PATH GUIDING

There are a great amount of global path guiding methods each of them with their level of complexity. One of the aims of this ETA is to have it be as simple as possible but still safe, for this reason it has been found that the best approach is to implement the A* algorithm. [45] [46] [47]

The A* algorithm is a widely used method in robotics that finds the optimal path from the robot's location to its desired destination. To achieve this, the navigation environment is represented in a graph, using nodes and edges. Nodes correspond to specific locations, so coordinates on the map and edges represent the possible paths between these nodes, so walkable areas. With this information the A* algorithm then calculates the cost to reach the goal node from the start node, and the path with the lowest cost is considered the optimal. This cost can be represented in different ways, such as distance, time or energy consumption and the key functions to calculate it are the following.

$$g(n) = \ cumulative\ cost\ from\ the\ start\ node\ to\ the\ current\ node$$

$$h(n) = heuristic\ estimate\ of\ the\ cost\ to\ reach\ the\ goal\ node\ from\ the\ current\ node$$

The total estimated cost of the path is represented in equation 37.

[45] Oxford (retrieved in 2024).
[46] Cornell University (2007).
[47] Stanford University (retrieved in 2024).

$$f(n) = g(n) + h(n) \quad (37)$$

The function g(n) allows the algorithm to correct possible error committed by the estimation and adapt to the changes that occur in a dynamic environment.

$$g(n) = g(n_{parent}) + cost(n_{parent}, n) \quad (38)$$

In this equation n_{parent} is the previous node and n is the next, so the equation 28 is updated for every new node. But the most important function out of these is h(n), the heuristic function. This function can take different forms depending on the path guiding necessities, the most common in robotics is to use the distance as cost and to implement the Euclidean distance equation.

$$h(n) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (39)$$

This is the simplest heuristic of all, because it assumes that the path between two nodes is a straight line. Although this isn't entirely true, since there can be obstacle in the way or a curve between the nodes, the estimation is sufficiently robust for this ETA. This is because since a line will always be the shortest path, the equation never overestimates the cost and therefore never disregards a shorter path, one of the main requirements for a valid heuristic function. To make this more reliable the nodes can be set closer to each other so that the real path is closer to a straight line.

In addition, the A* algorithm has a priority queue, which makes it ideal for real time applications since it allows it to be extremely efficient and fast. The priority queue organizes and processes the nodes based on their cost estimates, so that the most promising nodes (lower cost) are analyzed first, reducing the number of nodes that need to be examined and therefore reducing the algorithms response time.

3.4.2 OBSTACLE AVOIDANCE

For the obstacle avoidance a simple algorithm will be implemented, the first step for this method is to not only identify if there is an obstacle but to identify if that obstacle is in the user's path. To achieve this, the ETA will create a vector that goes through the robot's current position and the next waypoint determined in the A* algorithm. This vector represents the path that the user will take, therefore if any of the lidar points that are less than the range (detected an obstacle) are part of that vector the obstacle is in the way and obstacle avoidance must start. The following equations represent this explanation mathematically.

$$V_{path} = (x_{next\ waypoint} - x_{current\ position}, y_{next\ waypoint} - y_{current\ position}) \quad (40)$$

$$V_{cluster} = (x_{cluster\ point} - x_{current\ position}, y_{cluster\ point} - y_{current\ position}) \quad (41)$$

Both of these vectors form a line, therefore, to see if the obstacle is on the path or not, we must see if these vectors are colinear. To check for collinearity the cross product can be done, and if the result is zero it confirms they are collinear.

$$V_{path} * V_{cluster} = (x_{next\ waypoint} - x_{current\ position}) * (x_{cluster\ point} - x_{current\ position}) - (y_{next\ waypoint} - y_{current\ position}) * (y_{cluster\ point} - y_{current\ position}) \quad (42)$$

The next step is to determine whether that obstacle should be avoided by turning to the right or by turning to the left. If the cluster center is to the right of the vector, then the obstacle is more to the right and the robot should turn left, and if the center is to the left of the vector, then the obstacle is more to the left and the robot should turn right. To do this equation X is sufficient, since the turn depends on the result being positive or negative.

$$if \begin{cases} V_{path} * V_{cluster} > 0, & cluster\ center\ to\ the\ left \\ V_{path} * V_{cluster} < 0, & cluster\ center\ to\ the\ right \end{cases} \quad (43)$$

Once all of this information has been gathered the actual avoidance algorithm can start, but these previous steps are necessary to make this method safer and less complex. For the actual avoidance simplicity is key, because the less computational calculations needed the less errors the ETA will make. For this reason, there will be no differentiation between static and moving obstacles, and the robot will evade every obstacle with as sharp of an angle as it can at a constant speed. This decision has also been made due to the fact that none of the obstacles that the user will encounter move at high speeds, since it will mainly be other pedestrians walking. Therefore, there is no need to apply complex algorithms that would for example be used for autonomous driving. The avoidance will be done by adding two temporary waypoints that overwrite the waypoints created from the A* algorithm. The first waypoint will be added in front and to the side of the obstacle and the second waypoint will be added behind the obstacle in the path, so that the user is redirected. The following figure shows a graphic representation of this concept.

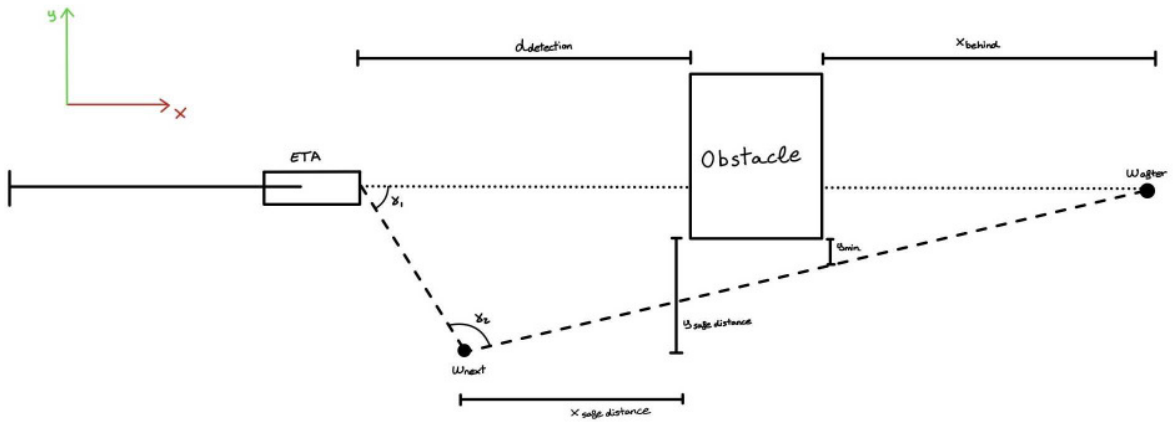


Figure 43: Obstacle avoidance representation. Source: Own elaboration (2024).

To correctly position the first waypoint two points will be used from the cluster, the leftmost and rightmost. Once these points are known the waypoint is calculated, by applying two equations and using as reference the robot's coordinate system.

If it was determined that the robot should turn right the rightmost point will be used and on the contrary if it should turn left the leftmost will be used.

$$y_{side\ waypoint\ right} = y_{rightmost\ point} - y_{safe\ distance} \quad (44)$$

$$y_{side\ waypoint\ left} = y_{leftmost\ point} + y_{safe\ distance} \quad (45)$$

For the x point it does not matter whether the robot turns to the right or to the left, the point used will be the one closest to the user to make sure it is correctly avoided.

$$x_{side\ waypoint} = x_{closest\ point} - x_{safe\ distance} \quad (46)$$

The second waypoint will be positioned at a certain distance behind the obstacle and at the same y than the ETA when the obstacle was first detected.

$$x_{behind\ waypoint} = d_{detection} + x_{behind} \quad (47)$$

The safe distances, as well as the x_{behind} will depend on the ETA's maximum turn angle and the y_{min}. So, the waypoints will be constricted by the following three inequalities.

$$\gamma_1 \leq \alpha_{max} \quad (48)$$

$$\gamma_2 \leq \alpha_{max} \tag{49}$$

$$y_{min} > \frac{user's\ maximum\ body\ width}{2} \tag{50}$$

With α_{max} being the ETA's maximum turn angle and the y_{min} restriction will make sure that the obstacle is surpassed so that the individual never collides with an obstacle, since its width is larger than the robot's. The relationship between angles and distances are the following.

$$\gamma_1 = \tan^{-1}\frac{y_{side\ waypoint} - y_{ETA}}{x_{side\ waypoint} - x_{ETA}} \tag{51}$$

$$\gamma_2 = \tan^{-1}\frac{y_{behind\ waypoint} - y_{side\ waypoint}}{x_{behind\ waypoint} - x_{side\ waypoint}} \tag{52}$$

In addition, the y_{min} will always be applied even whether an obstacle is in the user's direct path or not, since it can still cause a collision. This obstacle avoidance method will also be implemented if the sensor s_4 detects an obstacle overhead in the path to make sure all possible collisions are avoided.

3.5 CONCLUSIONS

With the end of this chapter the theoretical model of the ETA has been completed, with the next step being the simulation. The robot proposed can guide a visually impaired individual safely from point A to point B using A* algorithm for path finding and an obstacle avoidance algorithm. This obstacle avoidance has as inputs a 2D LiDAR sensor for waist level objects and an ultrasonic sensor for overhead obstacles. Since its main use is in outdoor urban environments it has also been designed to detect whether a traffic light is red or green, using an RGB camera. In addition, it is able to detect upward and downward staircases using three ultrasonic sensors strategically positioned to calculate the possible rise and thread. Furthermore, the system has been designed to be as small as possible, to achieve the objective of portability.

The main conclusion reached from this chapter is that the ETA has been designed to be as simple as possible, both the physical design and the technological design. The algorithms implemented all have one common characteristic and that is their low complexity and computational power needed. This has allowed for the creation of an innovative simple solution to the problem of navigation for the visually impaired.

Chapter 4. SIMULATION

In this chapter the ETA's simulation will be done using an advanced robot simulation software called Webots. This is a 3D program that provides a virtual environment for testing robot's physics, as well as all kinds of sensors and actuators. Webots also allows the user to code in different languages and in this case, Python will be used. For an overview of how the software works, it is divided into worlds and controllers. The worlds are the actual virtual environment and its components, and the controllers is the python code that is modified to tell the robot what to do and what not. The choice to use Webots came mainly for two reasons, the first being that you can create your own robot and the second being that all of the sensors needed for this simulation are supported by the program. This chapter will be divided into 5 sections: Model and world creation, stair simulation, traffic sign simulation, path guiding and obstacle avoidance simulation and finally conclusions. The simulations will show by parts, but it is all coded in the same controller so that all of the functions work together.

4.1 MODEL AND WORLD CREATION

The first step is to build the ETA and in order to do so another resource was used called OnShape, which is a cloud-base CAD software that provides tools for product design and development. This the 3D model of the wheel and bar were created and then exported into Webots, where they were joint.

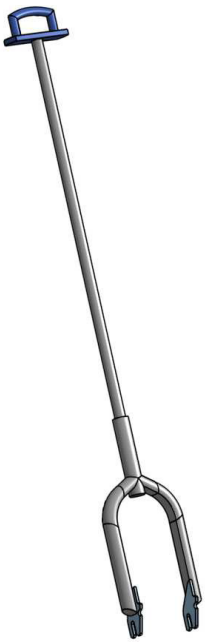


Figure 44: ETA's bar and handle represented in OnShape. Source: Own elaboration (2024).

Figure 45: ETA's wheel represented in OnShape. Source: Own elaboration (2024).

For the actual Webots representation there is still an issue because although, it has a great deal of advantages it has one very big disadvantage and that is that it doesn't allow for the simulation of walking humans. This is obviously a very big part of the project, which is why a work around has been found. The humans are simulated as a long triangle with the height and width of an average human and at the base they have to wheels to be able to move. In addition, the ETA has been joint to this figure at the height the visually impaired individual would hold the device. This has allowed the simulation to remain as close to reality as possible. The following image shows the robot in blue and the human lookalike in white.

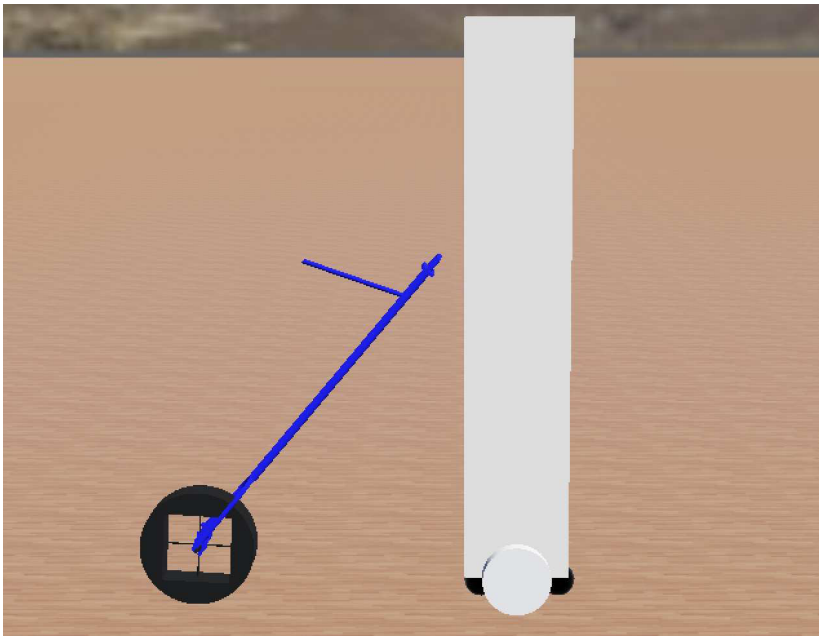


Figure 46: ETA and human represented in Webots. Source: Own elaboration (2024).

Once the model is created the different sensors are added according to Chapter 3's drawings.

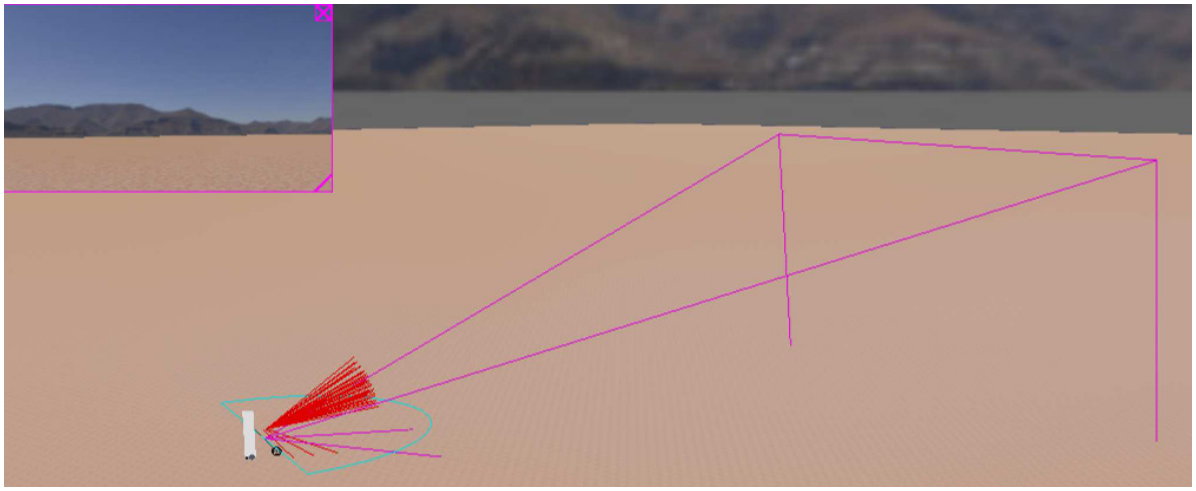


Figure 47: Sensor's fields of view in Webots. Source: Own elaboration (2024).

In Figure 40 the lines of the different sensor's fields of view are represented, where the red are the ultrasonic sensors, the blue the 2D LiDAR and the pink the RGB camera. In addition, in the top left corner the images captured by the camera are shown.

The last step before being able to simulate the ETA's functions is to create a world that resembles real-life situations the robot will encounter. This is relatively easy because Webots has thousands of already built in obstacles that can be used. To simulate all of the cases for this project, a set of upward stairs and downward stairs is needed, a crosswalk with a traffic light and sidewalks with obstacles. All of this has been integrated into one world called City, which is shown in the following images.

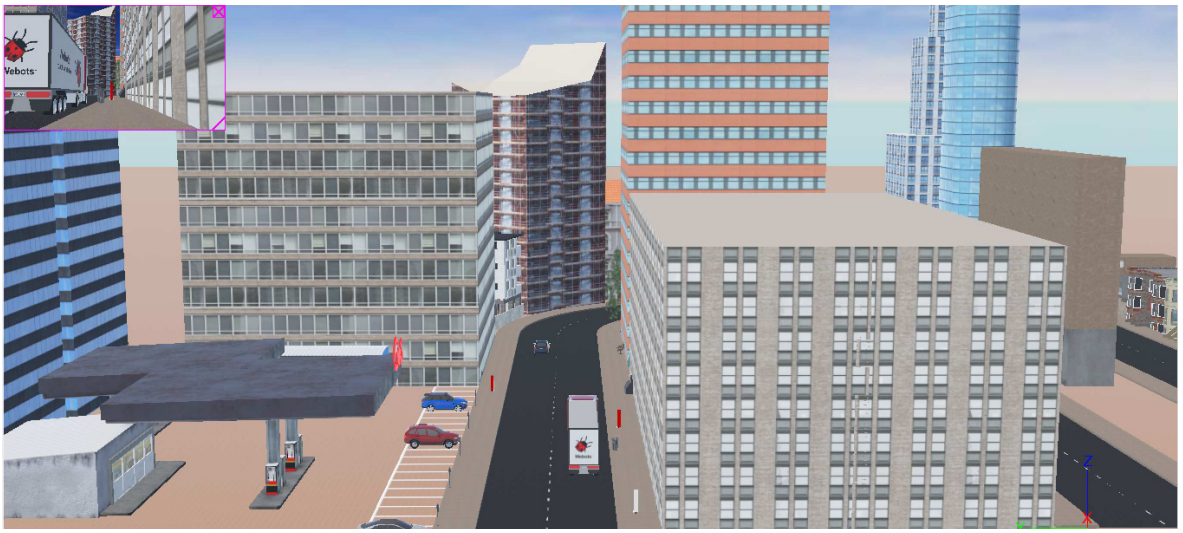


Figure 48: Overview of the City Webots World. Source: Own elaboration (2024).

Figure 49: Traffic light and crosswalk in the City World. Source: Own elaboration (2024).

Figure 50: Up and down staircase in the City World. Source: Own elaboration (2024).

4.2 STAIR SIMULATION

The first simulation done will be the staircase, starting with upward stairs and then downward stairs. The code is based on the mathematical model developed in Chapter 3

For both upward and downward staircase the user will receive two different feedbacks. The first message communicates to the individual the distance from the end of the wheel to the first step at the moment it is detected. The second message alerts the user that the front of the wheel has come in contact with the first step. In addition, the message will say whether that stair is an upward staircase or a downward one.

The sequential timing has also been implemented, with the red and green rays representing the sensor that is working and the grey rays representing the sensors that are not working.

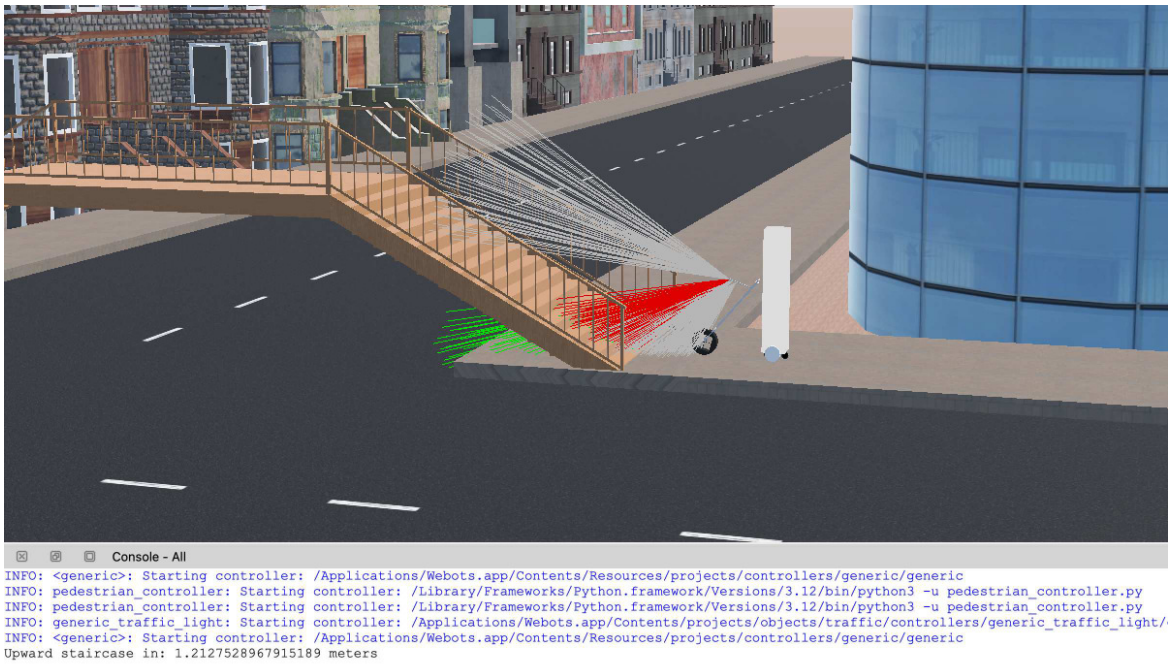


Figure 51: First upward staircase detection in Webots. Source: Own elaboration (2024).

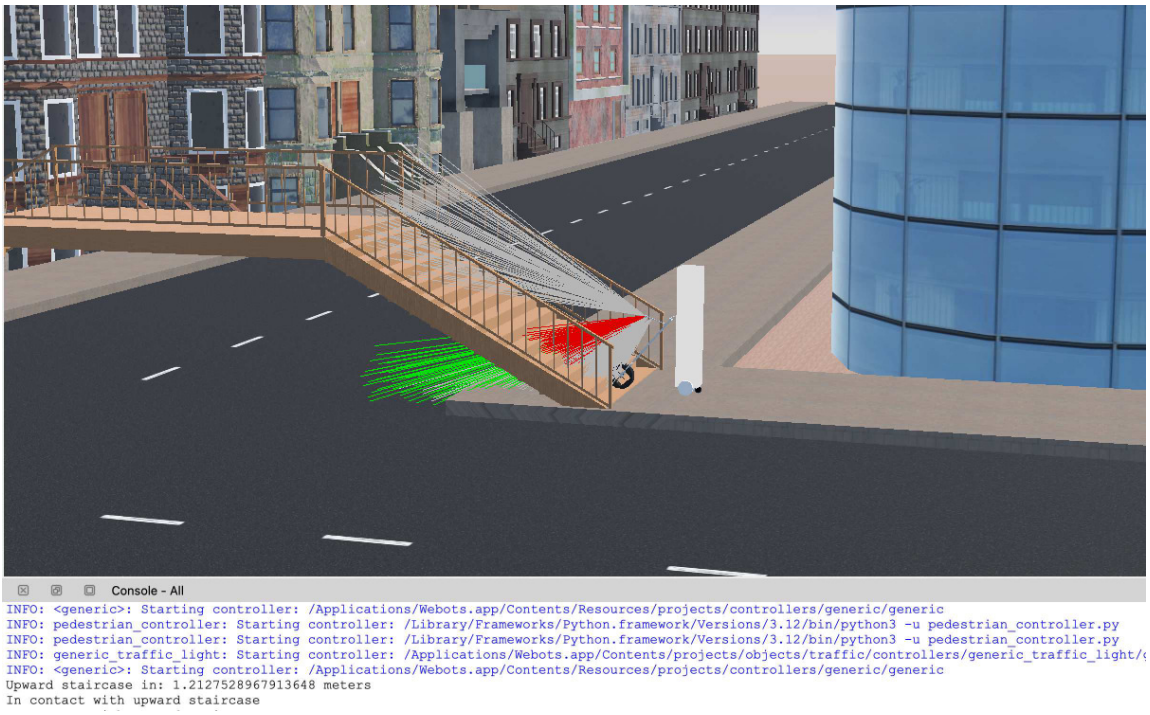


Figure 52: ETA In contact with upward staircase. Source: Own elaboration (2024).

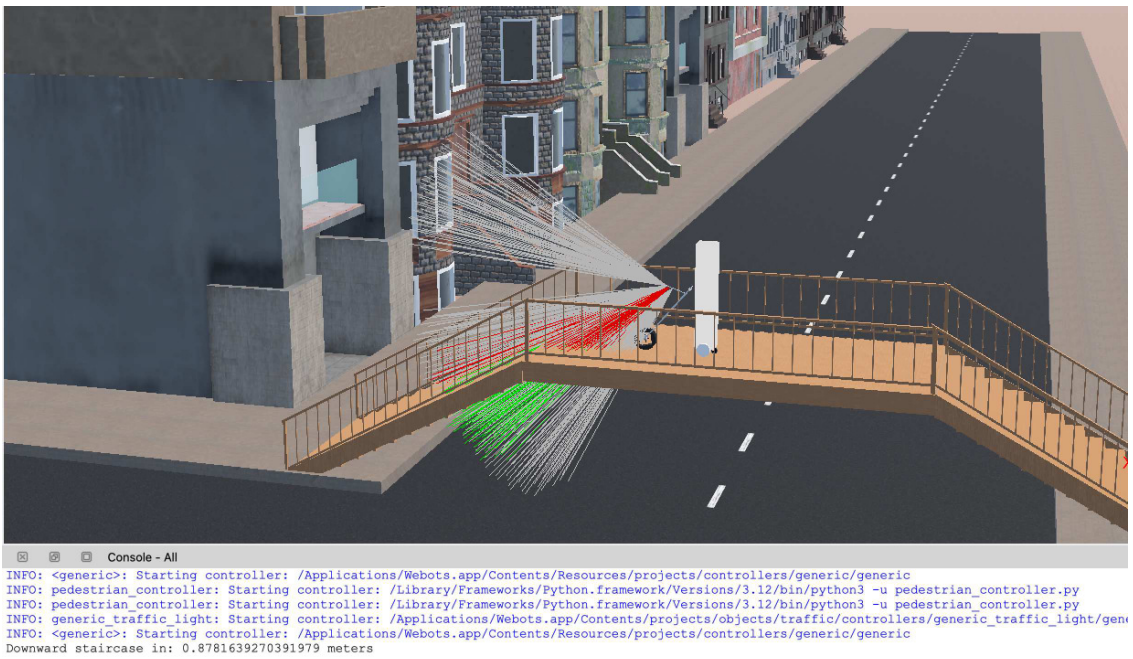


Figure 53: First downward staircase detection. Source: Own elaboration (2024).

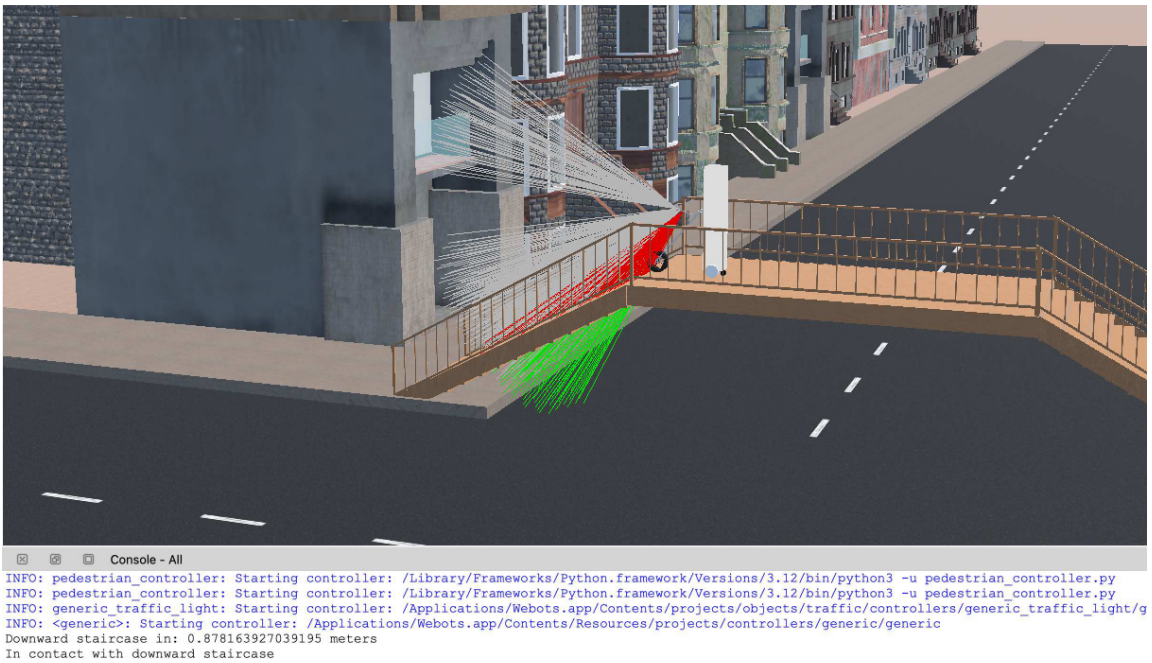


Figure 54: ETA in contact with upward staircase. Source: Own elaboration (2024).

As can be seen in Figure 44 the ETA can detect upward staircases from a distance of 1.213 meters and from Figure 45, we see that it can detect downward staircases form a distance of 0.878 meters. Although these distances are not particularly long, they are still considered safe since the stairs can still be felt with the actual ETA. So, there is a first layer of security, which is the first voice message and a second layer which would be the feeling of the robot stopping due to upward stairs or falling down due to downward ones. In addition, while the user is feeling the obstacle through the ETA another message confirms that indeed what they are feeling is the staircase and not another object.

Obviously due to the constraint of not having a real human but another robot holding the device the staircase simulation is limited since we are not able to simulate when the individual is on the staircase. Even though this is a slight problem since the primary goal was to detect the stairs while walking towards them, we can consider the simulation as a success. Furthermore, in the odd case that it didn't work once the user was on the stairs this would not cause a safety issue, since there would be no possibility of the person falling down. When going up the user would step higher than usual and when going down the user would step harder than usual, which causes no safety risks.

4.3 TRAFFIC LIGHT SIMULATION

For this simulation, the first and most important step is to figure out the values of green and red on the HSV color wheel. To start the ideal values will be used, and based on them a series of iterations and simulations will be done to find the correct value for the ETA. These values vary due to light conditions, and the RGB camera used and another fact to take into account is that since the Webots software is a simulation it could also cause variations from real life.

To be able to work with HSV values Python has a special library called OpenCV, however this library uses a different scale than the theory.

	Hue	**Saturation**	**Value**
Theoretical Scale	0º - 360º	0% - 100%	0% - 100%
OpenCV Scale	0 - 179	0 - 255	0 - 255

Table 5: HSV theoretical scale vs HSV OpenCV scale. Source: Own elaboration (2024).

Therefore, before the iteration and testing process can start the theoretical values must be converted to the OpenCV scale.

	Hue	**Saturation**	**Value**
Red Value Scaled	0 – 18 / 165 – 179	179 - 255	140 - 255
Green Value Scaled	79 - 89	188 - 255	50 - 246

Table 6: Theoretical HSV values scaled to OpenCV. Source: Own elaboration (2024).

It must be highlighted that with this change the red Hue value has now two completely different possible ranges and has to be taken into account for the iteration process. The goal with this iteration is to obtain values that not only detect green and red from a 25-meter distance but also do not mistake other colors for them. Another main concern is the camera detecting red and green from other surfaces and mistaken it for a traffic light, to avoid this several measures have been applied. The first being that the waypoints as they are in real-life will be programmed to have extra information, in this case one waypoint will be set before the crosswalk to inform of the distance to it and a second one will be set after the crosswalk to confirm it has been passed. This second waypoint has also been added in case the traffic light turns red while the user is still crossing so that the robot doesn't stop, on the contrary alerts the user of the danger they are in. Furthermore, the camera will remain off until the first waypoint has been reached, not only will this help to not make mistakes, but it will also help to prolong battery life and reduce overall complexity. The second is the previously explained region of interest or ROI, where the camera will only focus its efforts on the top half of the image. The last measure is a contour algorithm, when a red or green color is detected, this function is run to see if the color shape

matches a circle (traditional traffic light) or a human shape. In addition, although not an explicitly added measure the HSV color system takes into account the brightens or lack of in a color, and since traffic lights are LEDs, they have a different HSV value than for example an average surface, which also helps reduce possible mistakes.

To then affirm or deny the state of the traffic light the number of pixels that meet the measures imposed are counted and if that number is higher than a certain threshold then the state can be confirmed. This threshold will again be decided based on the different iterations done.

To determine the pixel threshold and HSV values, a first test outside of the created City World is done. This test consists of two scenarios, one with a red traffic light and green wall behind it and another with a green traffic light and a red wall behind it. With these two tests, iterations of values will be done starting with the theoretical ones until the green and red count return is correct. In addition, the distance at which the camera can detect the traffic light color will also be tested and worked on by positioning the robot 25 meters away from the traffic light while doing the iterations.

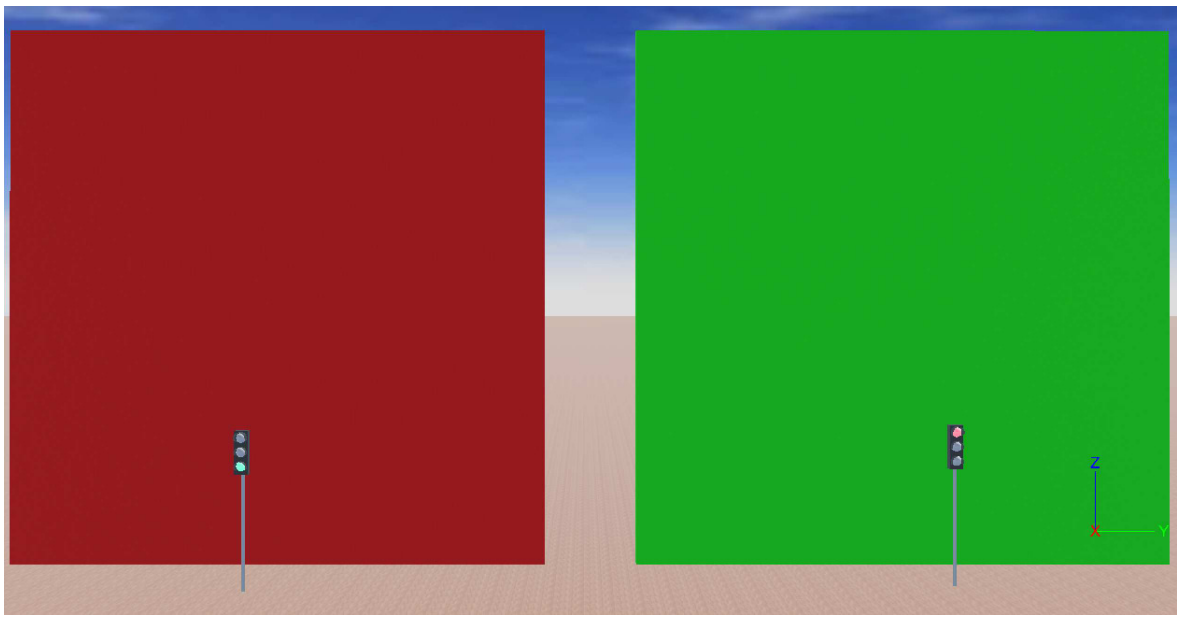


Figure 55: Traffic scenario to iterate theoretical values. Source: Own elaboration (2024).

After the iterations it was found that the optimal red and green HSV values were the following.

	Hue	Saturation	Value
Experimental Red Values	120 - 179	100 - 255	110 - 255
Experimental Green Values	35 - 95	100 - 255	100 - 255

Table 7: Experimental HSV values for red and green. Source: Own elaboration (2024).

In comparison to the theoretical values, these new values have a higher range, due to the fact that the lower limit is lower than in theory in most cases. Although there are some alterations, all of the ranges still make sense with the theoretical values and therefore the iterations can be considered a success. The following images show the results of these iterations, as well as the first pixel count for each scenario which will determine the pixel threshold since is always the minimum count value.

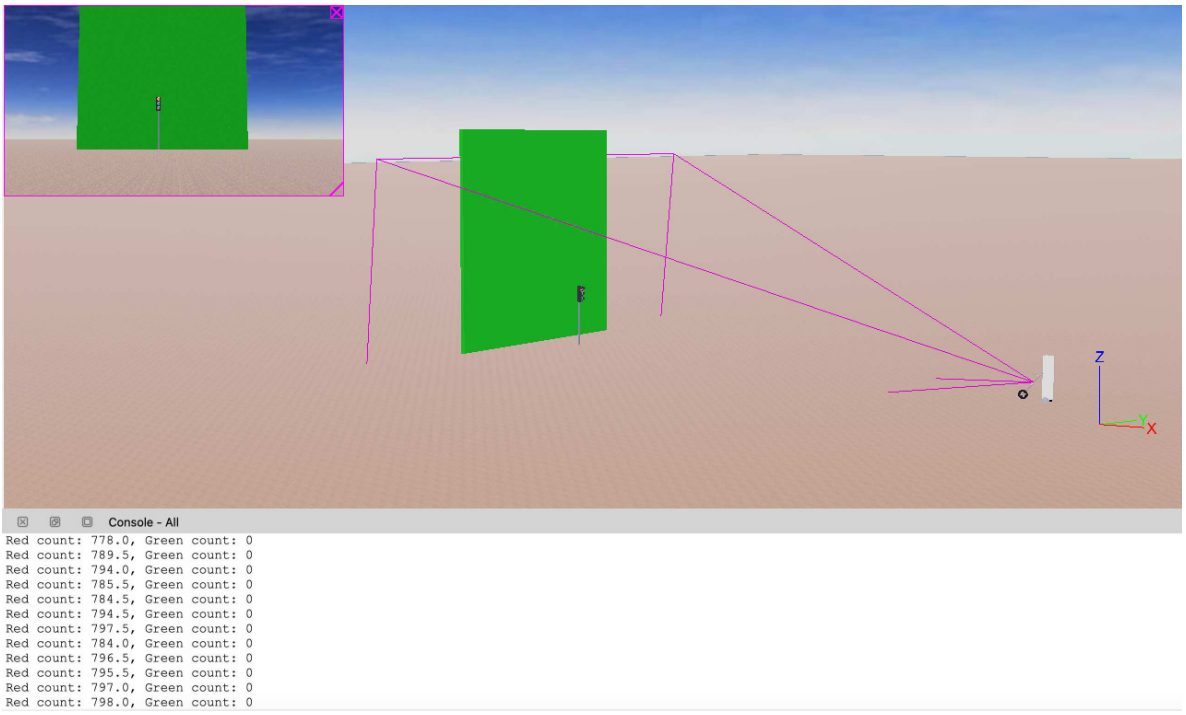


Figure 56: Red traffic light with green wall scenario. Source: Own elaboration (2024).

From the previous figure we see that the ETA functions correctly by only detecting the color from the red traffic light.

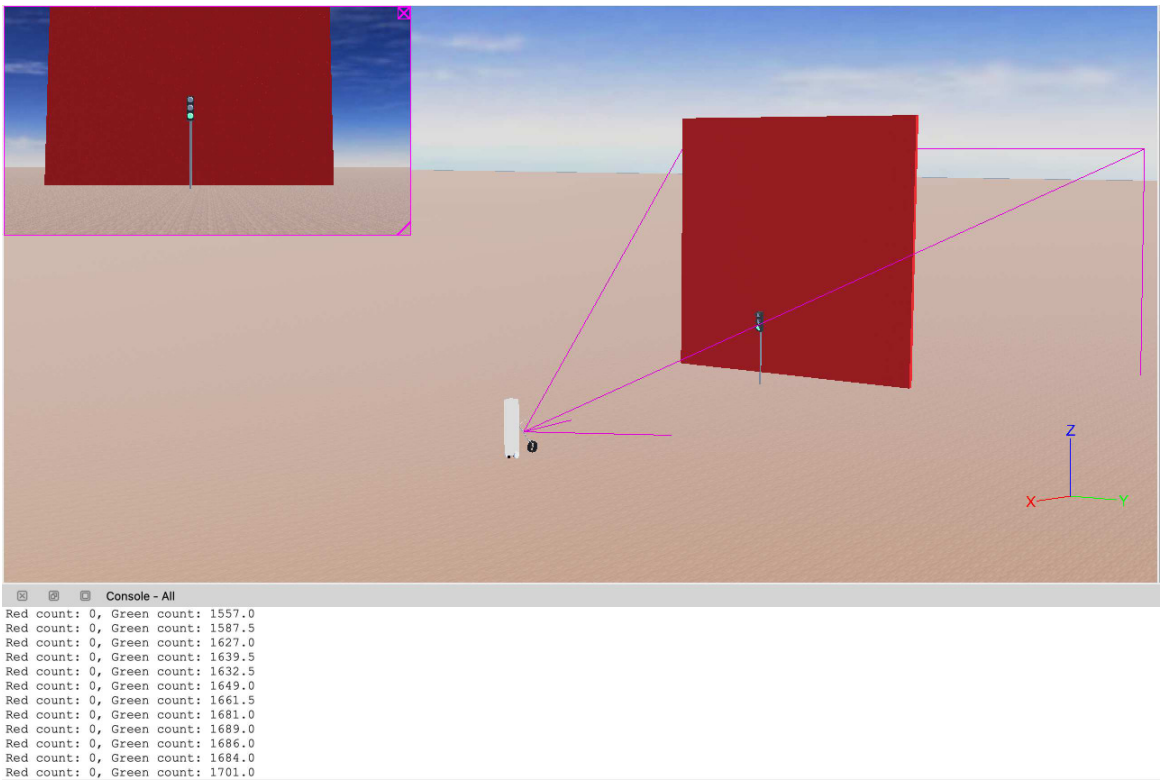


Figure 57: Green traffic light with red wall scenario. Source: Own elaboration (2024).

From Figure 50 we see again that the ETA functions correctly by only detecting the green from the traffic light.

With the values of the green and red pixel count from the green and red traffic light simulation scenarios respectively, the following graph was made to help determine the minimum count threshold. In addition, the graph will be used to determine the distances from which the camera can detect the traffic light color, as well as the distance at which it loses range. This will be done using the robot's GPS location and the traffic light's known position in the map.

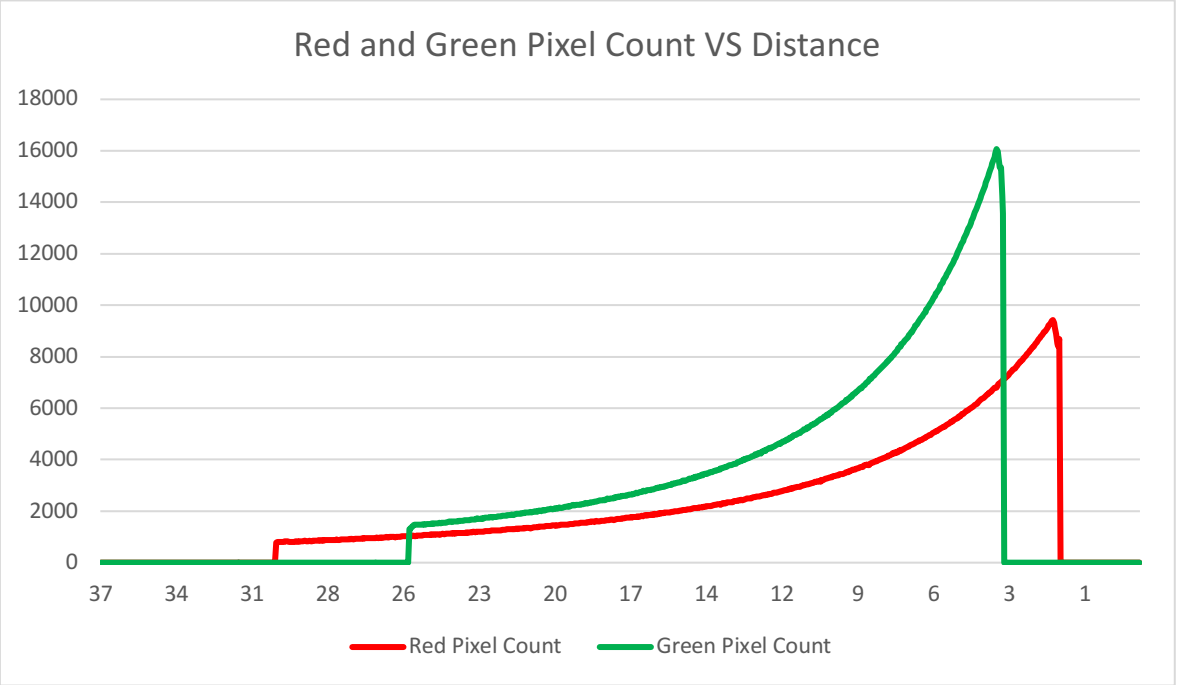


Figure 58: Red and green pixel counts vs distance graph. Source: Own elaboration (2024).

If we analyze the graph shown in Figure 51, we can see that for both the red and green the pixel count exponentially grows until the traffic light is no longer in range. With this information we now know that the threshold must be at least one pixel lower than the first registered count. If we look at Figure 49 and 50, we see that the first green pixel count is 1557 and the first red pixel count is 778, therefore we must choose the most constraining value which in this case is 778. The final threshold must be under this number, and since we have a very high margin of a confusion between red and green, some leeway can be given in case light conditions make these numbers fall. For this reason, the threshold for pixel count for both colors alike will be set at 720.

Furthermore, we see that when the light is red the camera has a much higher detection range, of more than 5 meters. This is not a worrying figure since the green still has an adequate range, starting at 26 meters and ending at almost 3 meters. These numbers satisfy the objectives, which were to detect all traffic lights form a 25-meter distance up until a 3 meter one.

With the detection working correctly we can start with the City World simulation, which as stated before will test that the waypoint works, and the ETA never stops in the middle of the sidewalk.

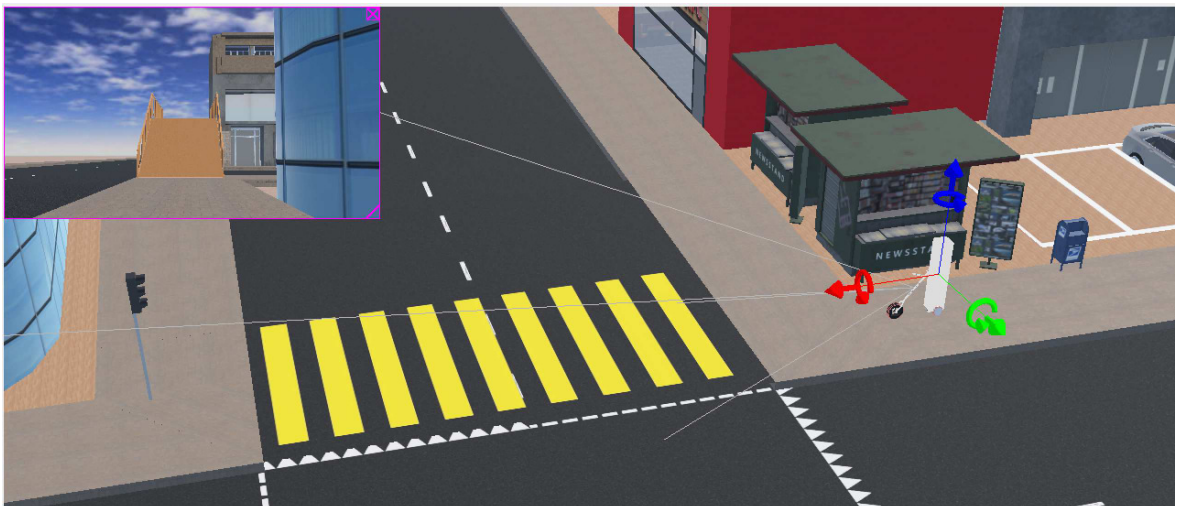


Figure 59: Traffic light detection before the first waypoint is reached. Source: Own elaboration (2024).

As can be seen in the previous image the user hasn't yet reached at the waypoint with the traffic light information and therefore the camera view seen in the top left corner is frozen. The fact that is frozen shows that the camera is off, since it will only turn on when it is time to detect a traffic light.

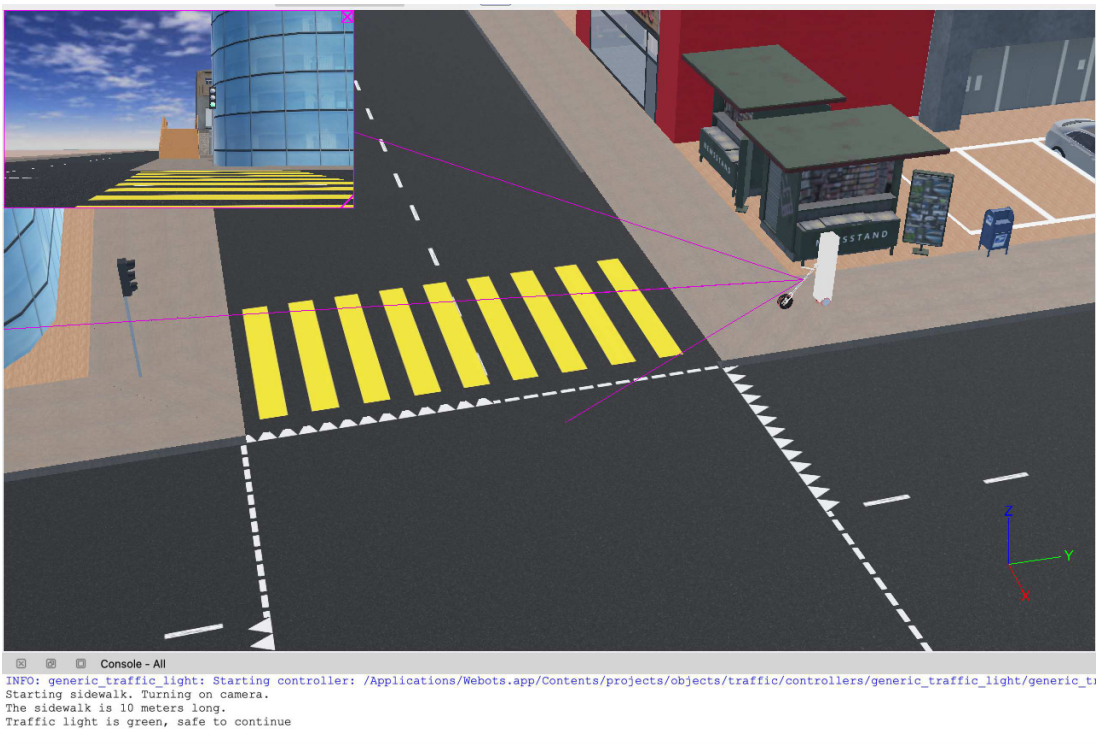


Figure 60: Green light detection after the first waypoint is reached. Source: Own elaboration (2024).

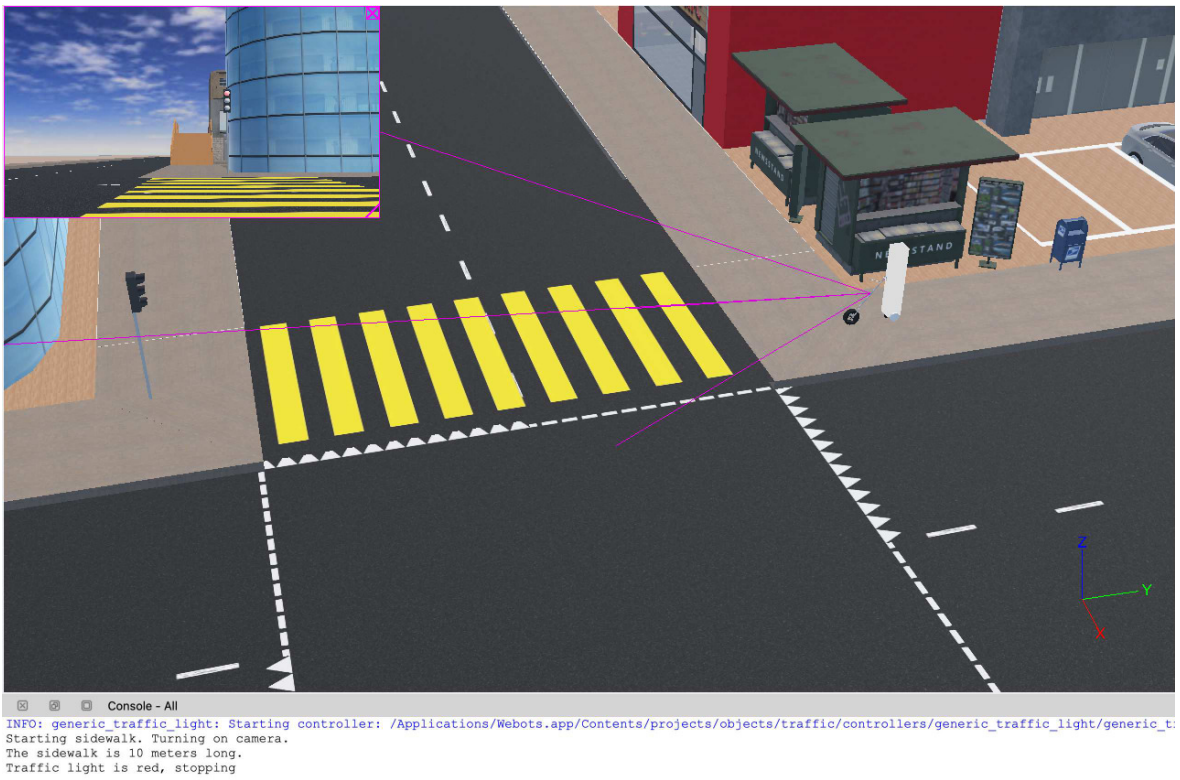


Figure 61: Red light detection after the first waypoint is reached. Source: Own elaboration (2024).

Once the waypoint is reached the camera turns on and the camera view starts showing the real image and not a frozen one. This is shown in Figure 54 and Figure 55, where the messages given to the user as feedback are also shown. Independently of the color once the waypoint is reached there is voice command that states the presence of a crosswalk, as well as its length. Then depending on the color, the user will be informed that it is safe to continue or that it is not, which will be accompanied by the ETA stopping. If the light is initially red, the device will wait until a green light is detected and will communicate the message “Traffic light is green, safe to continue” and continue advancing.

Figure 62: Traffic light detection after the second waypoint is reached. Source: Own elaboration (2024).

Once the second waypoint is reached the robot receives the information that the crosswalk has been surpassed, transmits it to the user via a voice command and then proceeds to turn the camera off.

The last part of this section is to make sure that even if the device detects a red traffic light it does not stop in the middle of the sidewalk. This is a very real issue that can happen when the time to cross is limited and could cause serious injuries to the visually impaired individual.

Figure 63: Red light detection in the middle of the crosswalk. Source: Own elaboration (2024).

From Figure 57 we can observe that when the light is detected red in the middle of the sidewalk an alert voice command is transmitted to make the user aware of the danger they are in. In this case the breaking command which would usually be emitted is overwritten so to make sure the ETA does not break and continues its course.

4.4 PATH GUIDING SIMULATION

In order to simulate path guiding using A* algorithm, waypoints and edges were manually added to the City World.

Figure 64: Location of the waypoints in City World. Source: Own elaboration (2024).

Since this was done manually not all of the city was mapped but only the necessary sections to show the correct functioning of the algorithm. The way it works is that the coordinates of one of the waypoints is inputted in the code and the device with the user's current GPS location calculates the most cost-efficient path using distances.

To show case how it works and to make sure it is correct three different cases where run. For each case the manual calculation will be done to confirm that indeed that would be the right path to take. For this it is necessary to also now the precise waypoint location which is shown in the following table.

Point	Coordinates
A	(-44.8, 12.6)
B	(-34.8, 12.6)
C	(-24.8, 12.6)
D	(-14.8, 12.6)
E	(-4.8, 12.6)
F	(5.2, 12.6)
G	(9.8, 12.6)
H	(19, 12)
I	(22.6, 10.7)
J	(26.1, 8.29)
K	(30.4, 3.28)
L	(32.5, -1.96)
M	(33.3, -6.1)
N	(33.3, -18.6)
O	(33.3, -30.6)
P	(33.3, -36.1)
Q	(33.3, -43.9)
R	(-12.5, 16.7)
S	(-10.8, 10.9)
T	(-10, 2.15)
U	(-10.5, -3.85)
V	(-10.5, -13.85)
W	(-10.5, -16.1)
X	(-1, -18.6)
Z	(1, -18.6)
AA	(12.52, -18.6)
AB	(22.52, -18.6)
AC	(31.4, -18.6)

Table 8: City World Webots waypoint coordinates. Source: Own elaboration (2024)

4.4.1 CASE 1

For the first case the ETA will start near point A and the goal was set to be waypoint Q or (33.3, -43.9) in the map's coordinate system.

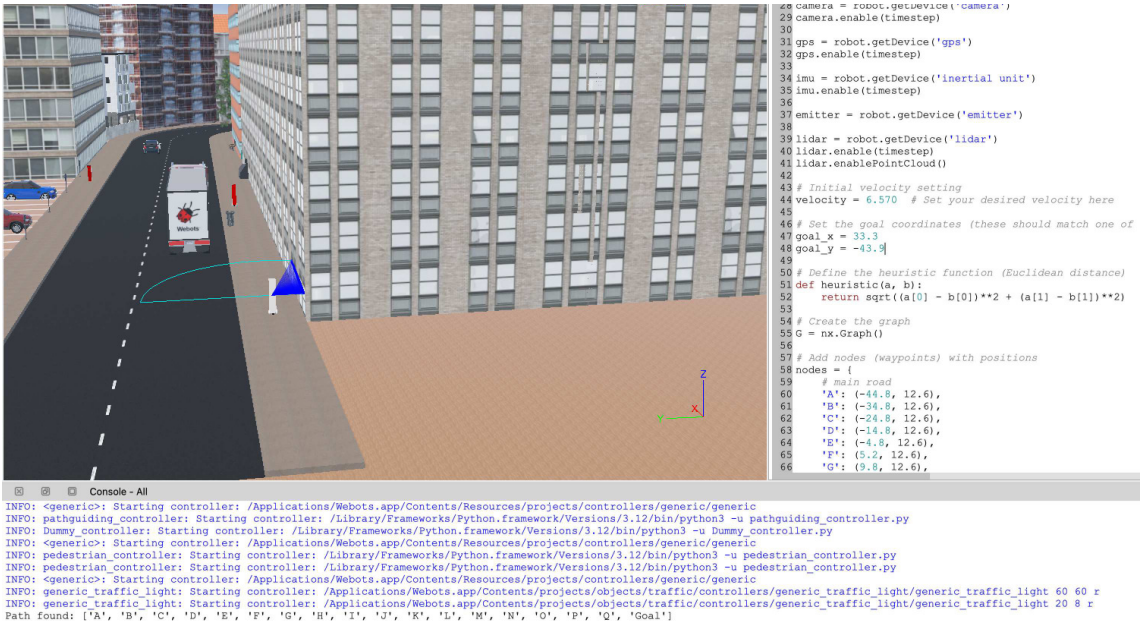


Figure 65: Path guiding Webots simulation Case 1. Source: Own elaboration (2024).

Figure 66: Representation of Case 1's path guiding solution. Source: Own elaboration (2024).

To check if the path {A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q / Goal} is optimal we need to calculate the cost of that path, as well as the one going through the shortcut. To do so not all of the edges' costs have to be calculated, since the equation used is the Euclidean distance which makes straight lines as edges. This means that for straight lines we can just use the first and last waypoint that forms them.

Following is the calculation of the total cost for the optimal path provided by the algorithm.

$$Cost\ (A,G) = \sqrt{(9.8 - (-44.8))^2 + (12.6 - 12.6)^2} = 54.6$$
$$Cost\ (G,H) = \sqrt{(19 - 9.8)^2 + (12 - 12.6)^2} = 9.22$$
$$Cost\ (H,I) = \sqrt{(22.6 - 19)^2 + (10.7 - 12)^2} = 3.83$$
$$Cost\ (I,J) = \sqrt{(26.1 - 22.6)^2 + (8.29 - 10.7)^2} = 4.25$$
$$Cost\ (J,K) = \sqrt{(30.4 - 26.1)^2 + (3.28 - 8.29)^2} = 6.60$$
$$Cost\ (K,L) = \sqrt{(32.5 - 30.4)^2 + (-1.96 - 3.28)^2} = 5.65$$
$$Cost\ (L,M) = \sqrt{(33.3 - 32.5)^2 + (-6.1 - (-1.96))^2} = 4.22$$
$$Cost\ (M,Q) = \sqrt{(33.3 - 33.3)^2 + (-43.9 - (-6.1)^2} = 37.8$$

$$Algorithm\ Path\ Total\ Cost = 126.17$$

Now we will do the calculation of the total cost if the device were to take the shortcut, we can reuse some of the values calculated above.

$$Cost\ (A,D) = \sqrt{(-14.8 - (-44.8))^2 + (12.6 - 12.6)^2} = 30$$
$$Cost\ (D,R) = \sqrt{(-12.5 - (-14.8))^2 + (16.7 - 12.6)^2} = 4.70$$
$$Cost\ (R,S) = \sqrt{(-10.8 - (-12.5))^2 + (10.9 - 16.7)^2} = 6.04$$
$$Cost\ (S,T) = \sqrt{(-10 - (-10.8))^2 + (2.15 - 10.9)^2} = 8.79$$
$$Cost\ (T,U) = \sqrt{(-10.5 - (-10))^2 + (-3.85 - 2.15)^2} = 6.02$$
$$Cost\ (U,W) = \sqrt{(-10.5 - (-10.5))^2 + (-16.1 - (-3.85))^2} = 12.25$$
$$Cost\ (W,X) = \sqrt{(-1 - (-10.5))^2 + (-18.6 - (-16.1))^2} = 9.80$$
$$Cost\ (X,N) = \sqrt{(33.3 - (-1))^2 + (-18.6 - (-18.6))^2} = 34.30$$
$$Cost\ (N,Q) = \sqrt{(33.3 - 33.3)^2 + (-43.9 - (-18.6))^2} = 25.3$$

$$Alternative\ Path\ Total\ Cost = 137.47$$

Since $Algorithm\ Path\ Total\ Cost < Alternative\ Path\ Total\ Cost$, the code was successful in finding the optimal path.

4.4.2 CASE 2

For the second case the ETA will start near point A and the goal was set to be waypoint AA or (12.52, -18.6) in the map's coordinate system.

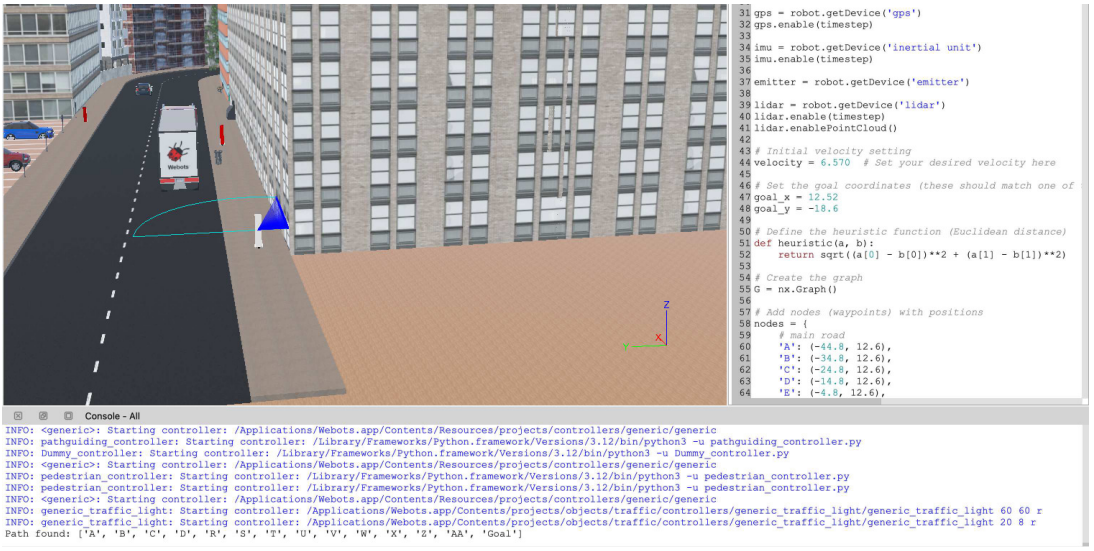


Figure 67: Path guiding Webots simulation Case 2. Source: Own elaboration (2024).

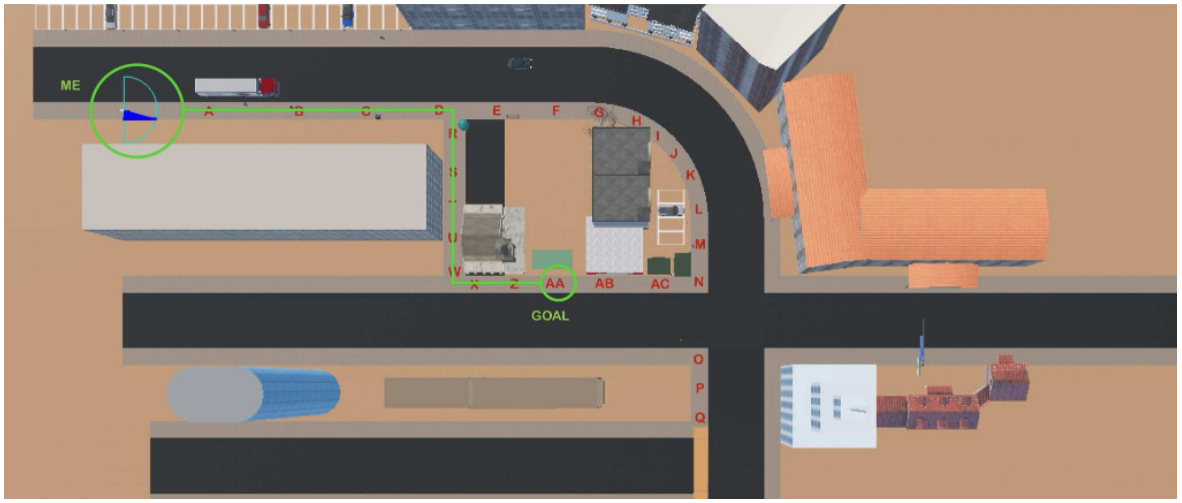


Figure 68: Representation of Case 2's path guiding solution. Source: Own elaboration (2024).

In this case the optimal path is a lot clearer and can be known just looking at the image since the shortcut allows the device to not have to go forward and then backwards. Therefore, without the need of calculating costs it can be confirmed that the code has worked correctly.

4.4.3 CASE 3

This last case is done to reaffirm that the A* algorithm is working and to test that the GPS start is also working as it should.

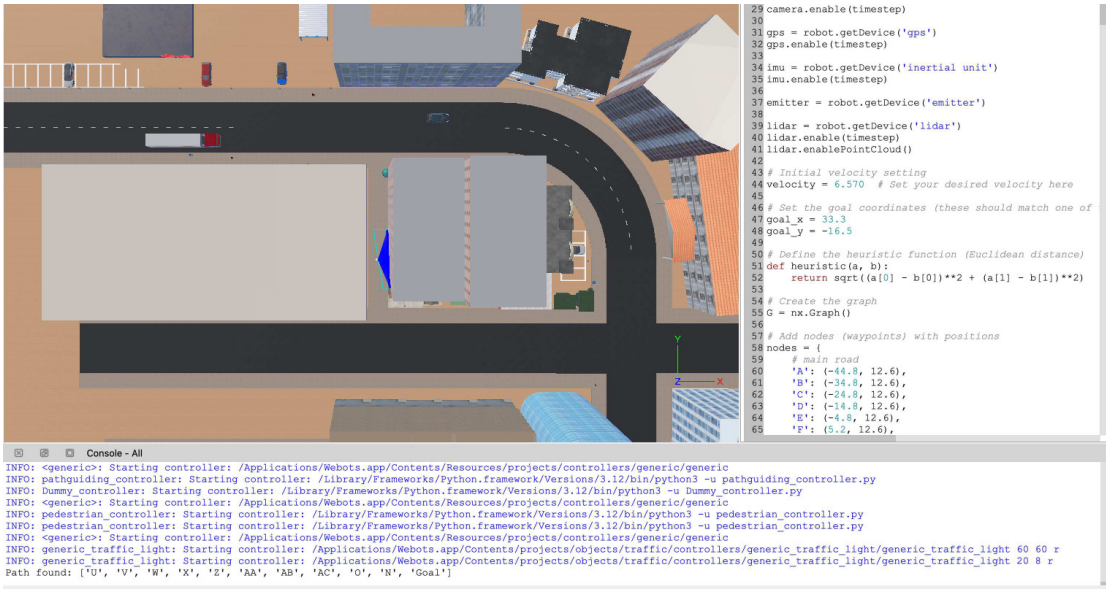


Figure 69: Path guiding Webots simulation Case 3. Source: Own elaboration (2024).

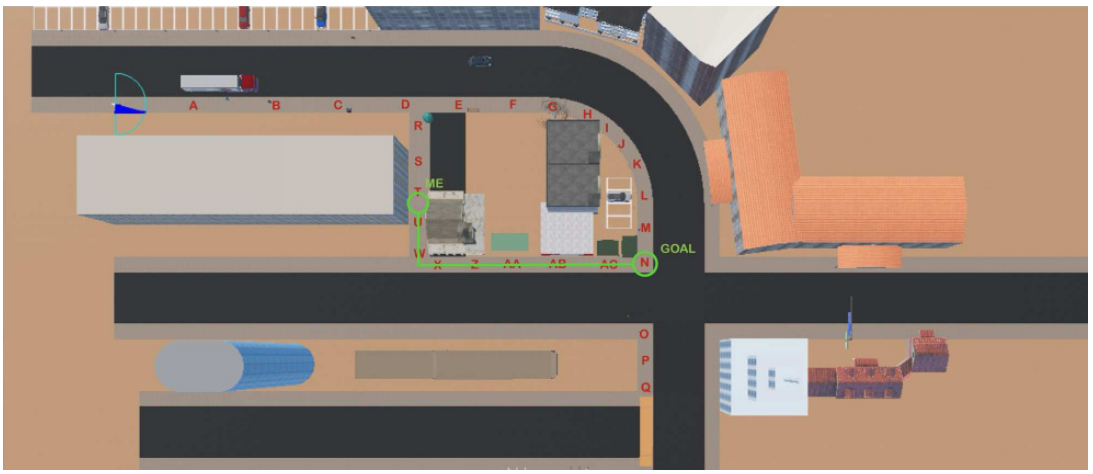


Figure 70: Representation of Case 3's path guiding solution. Source: Own elaboration (2024).

The optimal path is also intuitive in this case and from Figure 64 we confirm that the shorter route is to continue through the shortcut. This also demonstrates that the algorithm isn't just using the first waypoint, which would be A as the start but is using the ETA's GPS location.

4.5 OBSTACLE AVOIDANCE SIMULATION

The last section in this simulation chapter is obstacle avoidance, one of the ETA's most important functions. To make sure the logic implemented works correctly different scenarios will be done to test different functions, including overhead and low obstacles which are the ones outside the LiDAR's field of view. In addition, we will make sure the obstacle avoidance function only occurs when an obstacle is detected to be in the user's path, which as explained in the previous chapter will be done using the device's current location and the next waypoint in its path. We will also test that the ETA avoids the obstacle by turning in the right direction, meaning if the object is more to the left it should turn right and if its more to the right it should turn left. For all of the simulations the ETA will move at the average walking speed which is 4 km/h, and a test will be done at the end to see what the maximum speed that it could avoid is.

We will start by making sure that overhead and lower obstacle are also detected by the device. Then we will test if the robot can detect an obstacle in its way and if it can turn to the correct side, since these are the first steps in implementing a robust obstacle avoidance logic.

4.5.1 LOW OBSTACLE DETECTION

For obstacles that are lower than the 2D LiDAR's field of view, ultrasonic sensor number 1 will be used. Only this sensor will be used since it has a longer view and will therefore always detect the obstacles first. As can been seen in the next image this sensor can detect an obstacle from a 1.78-meter distance. In addition, the wheel will be the first to come into contact with the obstacle, and therefore if any of the obstacle avoidance logic were to fail this would work as a backup.

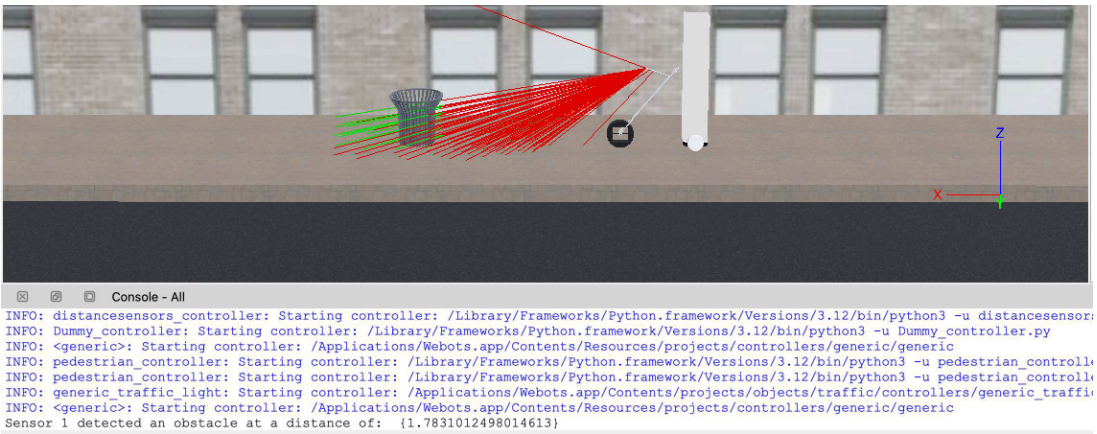


Figure 71: Low obstacle detection. Source: Own elaboration (2024).

4.5.2 OVERHEAD OBSTACLE DETECTION

For overhead obstacles ultrasonic sensor number 4 will be used. When this sensor detects an obstacle, the obstacles height is calculated and if it is lower than the user then the avoidance logic will be implemented. This height threshold will have leeway in case there are errors in the calculation, so that the user never runs into something.

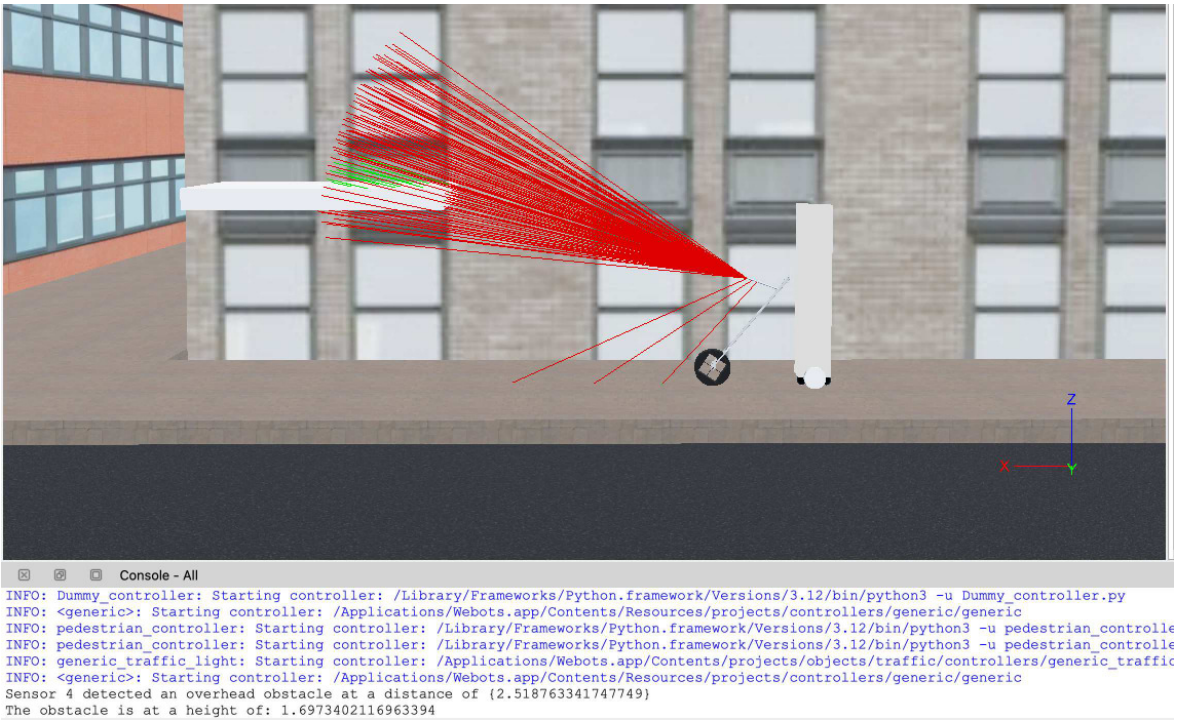


Figure 72: Overhead obstacle detection. Source: Own elaboration (2024).

4.5.3 DETECTION OF AN OBSTACLE IN THE PATH AND TURNING LOGIC

To test this, a block simulating a human will be strategically placed in different points of the device's field of view to see if the ETA returns it as in the path or not, and if it is in the path whether it will turn left or right.

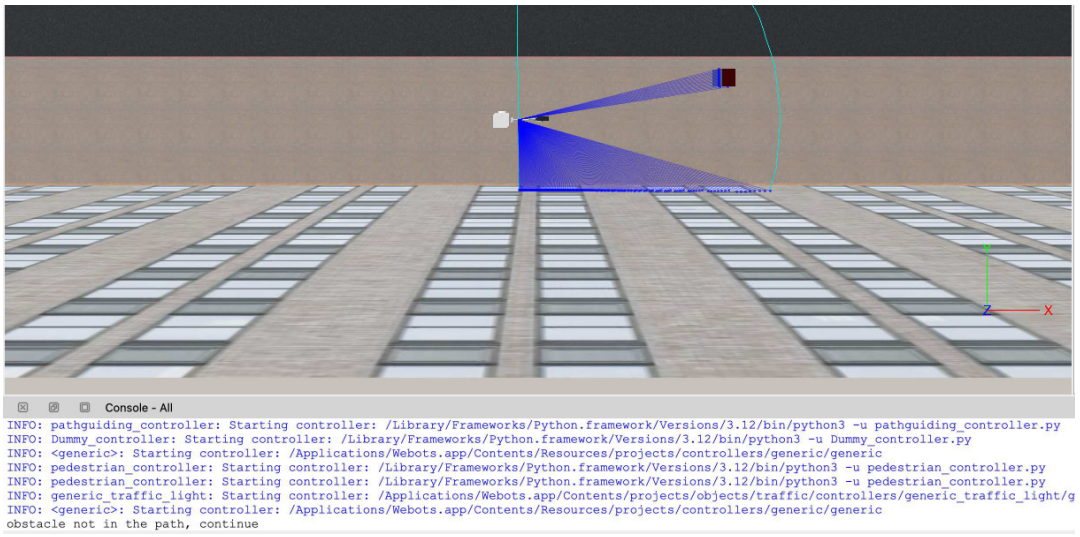


Figure 73: Obstacle to the left and not in the user's path. Source: Own elaboration (2024).

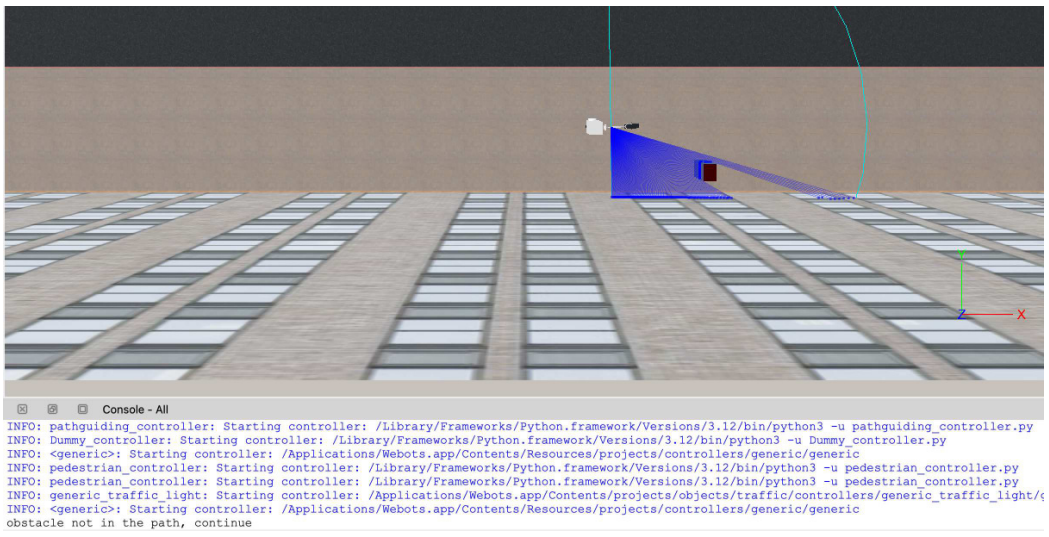


Figure 74: Obstacle to the right and not in the user's path. Source: Own elaboration (2024).

In Figure 57 and 58 the obstacle was placed as close to the user's path as possible without being actually in it. This was done to test the most extreme case and make sure the logic worked. As seen in the command output for both cases the return message is that

there is in fact an obstacle, but it is not in the way and therefore the ETA can continue as normal, which is exactly the outcome we were looking for.

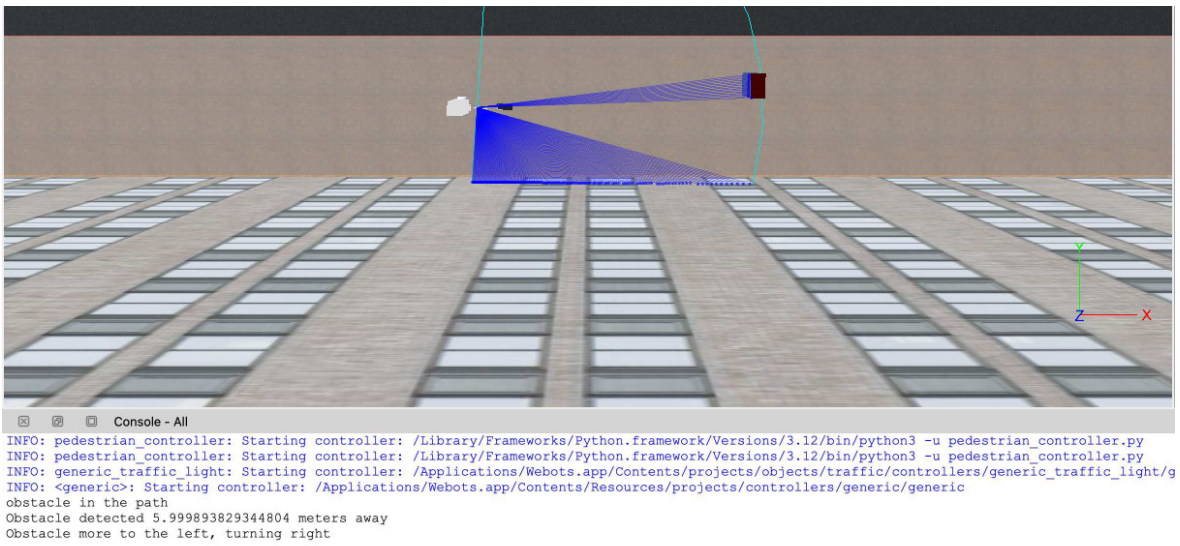


Figure 75: Obstacle to the left and in the user's path. Source: Own elaboration (2024).

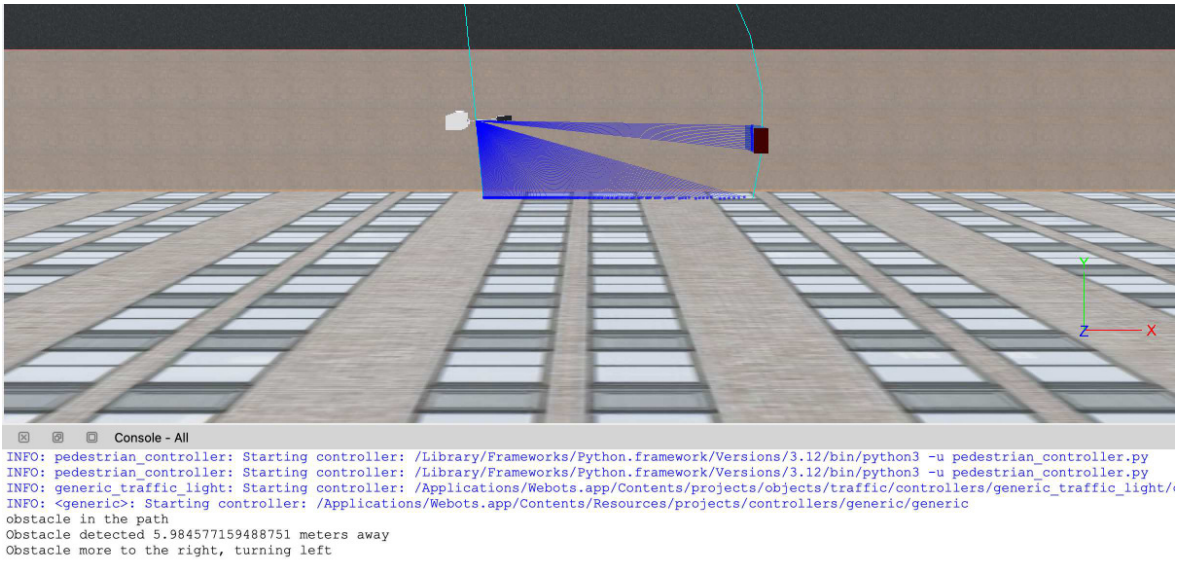


Figure 76: Obstacle to the right and in the user's path. Source: Own elaboration (2024).

Similarly, the obstacle has now been placed in the user's path and as close to the middle as possible to test the most extreme turning logic case. And again, as can be seen the robot behaves as expected, identifying the object in the path, as well as its position and turning towards the opposite direction.

4.5.4 WAYPOINT CREATION

For the waypoint creation which is the main obstacle avoidance method several scenarios will be tested to make sure the logic works as it should.

4.5.4.1 One obstacle

The first scenario is the simplest one, which will be the avoidance of one single obstacle in the user's path.

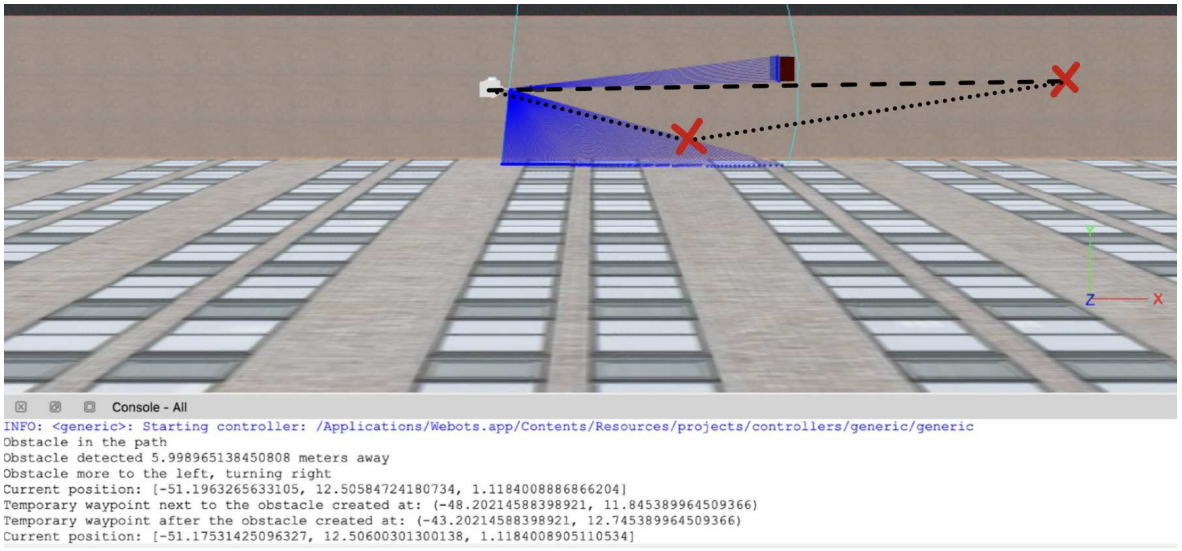


Figure 77: Obstacle to the left, waypoint creation. Source: Own elaboration (2024).

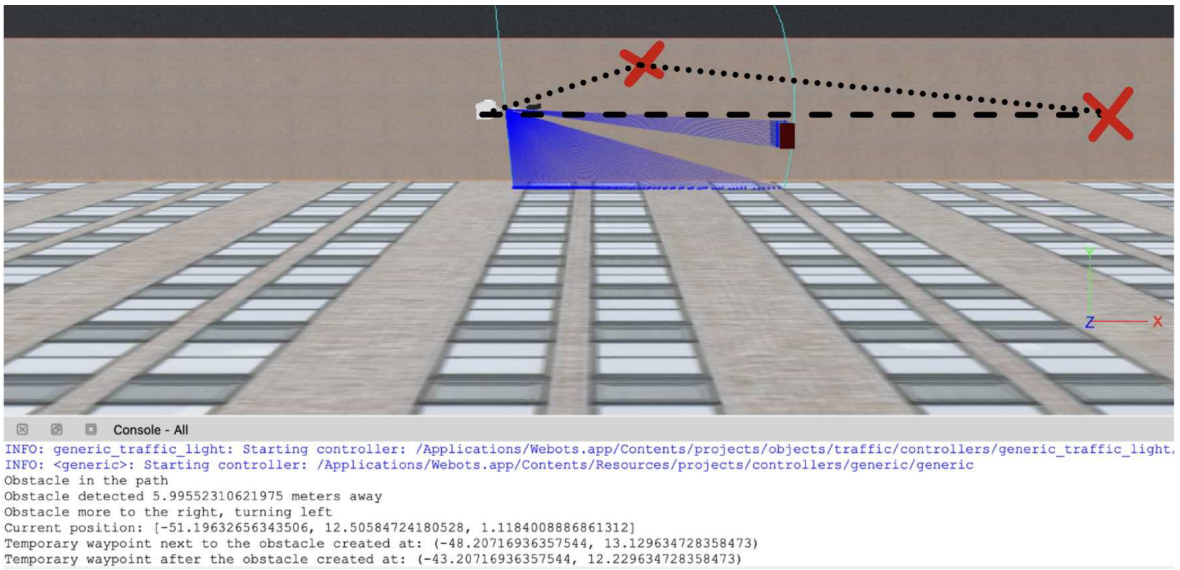


Figure 78: Obstacle to the right, waypoint creation. Source: Own elaboration (2024).

4.5.4.2 More than one obstacle

Since this will be used in urban environments it is essential that the ETA knows how to act when it encounters more than one obstacle. The first step with this is the clustering method, meaning if two different obstacles are so close to each other that the user cannot fit through them the robot will treat them as one. This function allows it so that the more than one obstacle logic is used less, which permits for a less complex overall avoidance. The case where they are not close to each other and the user could potentially fit through them will happen when the obstacles are at different distances, meaning that one of them will be encountered first and then the second. The issue with this is that the new waypoint created after obstacle 1 could potentially be where obstacle 2 is, therefore for this case we need to add an overwrite to the code. So, if the ETA detects a second obstacle the created waypoints will be updated for the position of this new object. This works because the system will only enter this process if the device can go through them.

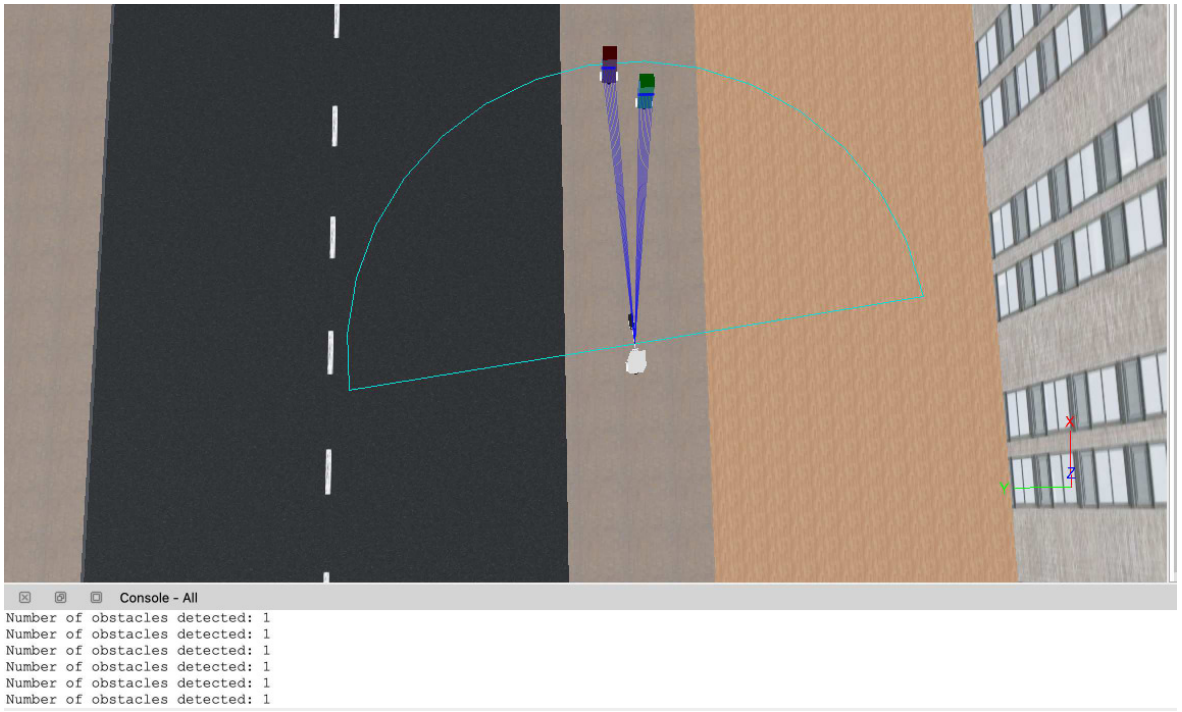


Figure 79: Two obstacle close to each other, returned as only one. Source: Own elaboration (2024).

Figure 75 shows the case of two “pedestrians” walking close to each other and since the blind individual could not fit in between them, the system returns them as only one.

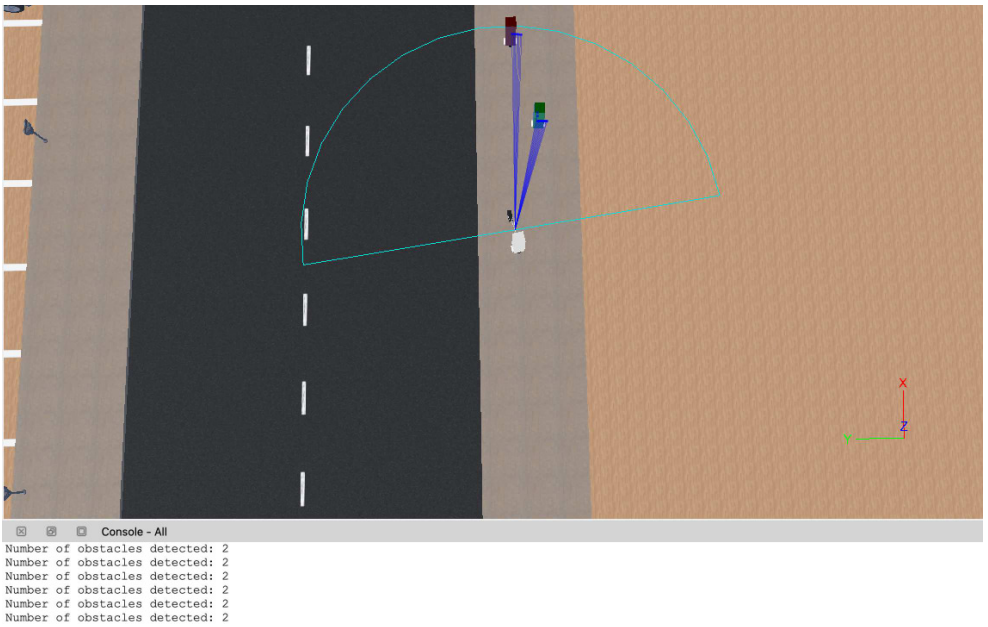


Figure 80: Two obstacles far from each, returned as two. Source: Own elaboration (2024).

Now we show the other case where the individual could potentially walk through them and therefore, they are detected as two obstacles. The next image shows this exact case and how the initial waypoints created are overwritten. The new ones are then calculated based on the robot's position, so in the following case it is more optimal to overwrite the waypoints to keep going to the left.

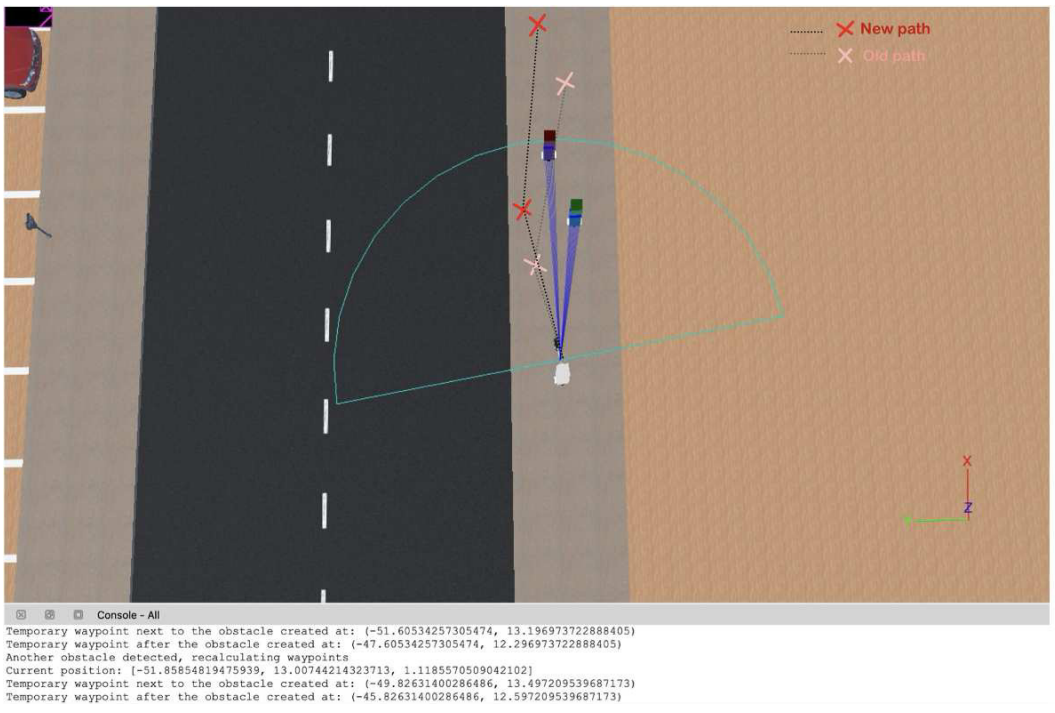


Figure 81: Waypoint recalculation for two obstacles. Source: Own elaboration (2024).

4.5.4.3 Maximum Speed an Obstacle can have

As stated previously, this test will be done with the ETA moving at a speed of 4 km/h, and the moving obstacle will be set in the center of the path. When it is in the center a default has been set so that the device turns towards the left. Doing the simulation will allow us to know in the worst-case scenario the maximum speed it can avoid, meaning if the obstacle was to one of the sites the velocity would be superior and if the visually impaired individual moved faster, it would also be bigger. Lastly, in most of the cases in real life the other pedestrian or bicycle would also try to avoid it.

Now, an issue when trying to test this was encountered, and that is that since Webots does not allow for human simulation when entering higher speeds, the block used becomes unstable and falls. Therefore, the ETA's limit can't actually be reached in this simulation. The human block falls once the incoming obstacle reaches a velocity of 10.98 km/h, what this tells us is that it could 100% avoid other humans walking and a person jogging. Obviously, this isn't ideal, but in reality, this speed will be much higher, since a human doesn't destabilize, and the ETA can turn up to 90º, which means that the moment it detects an incoming object it can make a complete turn of direction. For this reason, the fact that it could only be tested for lower speeds is not at all worrying since the actual logic of setting the waypoints correctly does work, which gives us no reason to believe that it wouldn't keep working for other speeds.

4.6 CONCLUSION

Once all of the tests have been done, it can be affirmed that the theory developed works in practice. The code implemented allows this device to be able to detect up and down staircases, detect traffic lights, guide the user through the optimal path, and avoid overhead, waist level and low obstacles. All of the functions worked exactly how intended and where able to be tested, except of course the maximum device speed. As stated previously this isn't an issue of the actual device but that the simulation didn't support real humans. Therefore, it can be concluded that the testing of the ETA has been a success.

Chapter 5. ECONOMIC ASPECTS

In this chapter an analysis of the economic aspects associated to producing and implementing the electronic travel aid for the visually impaired will be done. To start a brief market overview will be done and then the actual cost will be estimated, in order to see where this device falls in relation to current applications.

5.1 MARKET ANALYSIS

There are at least 2.2 billion people in the world who have some type of visual impairment, out of those 135 million people need some kind of aid and out of those 40 million are registered as being completely blind. This translates to a huge global market worth approximately 4.2 billion dollars in 2023 and is expected to reach 13.2 billion dollars by 20223 with a projected compound annual growth rate (CAGR) of 13.8%. [48] Although this market includes all types of assistive device for the blind and not only those focused on navigation it still gives us a broad idea of the importance of this market.

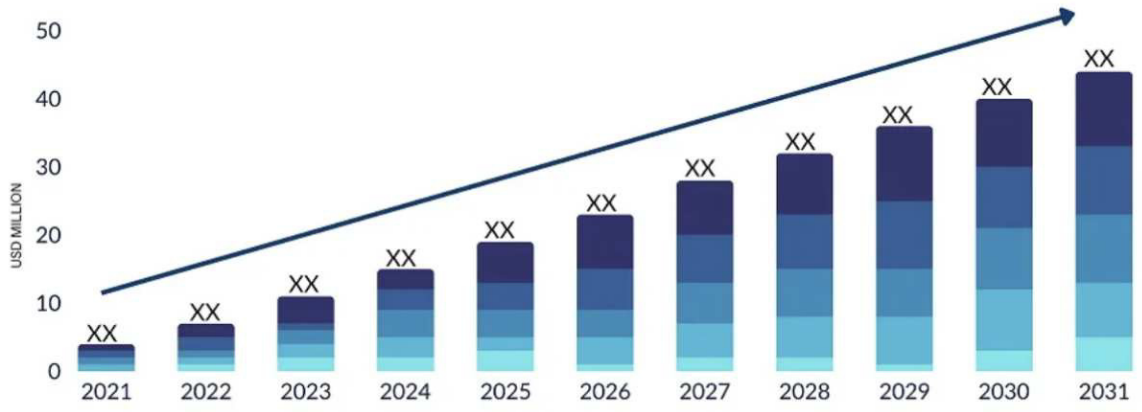


Figure 82: Assistive technologies for blind market growth. Source: Market Reports (2021).

This growth can be attributed mainly to two facts. The first is the quick progress of technology as a whole, for example with the development of machine learning or artificial intelligence. This has allowed assistive technologies to evolve by creating new innovative solutions, which in turn allows the market to keep growing. The second fact is the aging of the population due to higher rates of life expectancy. With aging comes deterioration which in a lot of cases causes vision-related disorders that could turn into partial or complete blindness. With life expectancy expected to only keep growing so will the assistive market with it.

[48] Verified Market Reports (retrieved in 2024).

As for the regional breakdown of the market, the most prosperous regions are North America and Europe. This is due to their advanced healthcare and education infrastructure, their older population and their large visually impaired population. Although these two continents dominate the market, especially North America with a share of 39%, the true growth is expected to happen in Asia. Therefore, the higher lucrative markets in the near future are those in Asia, especially in countries like India, where supportive government policies are starting to emerge. [49]

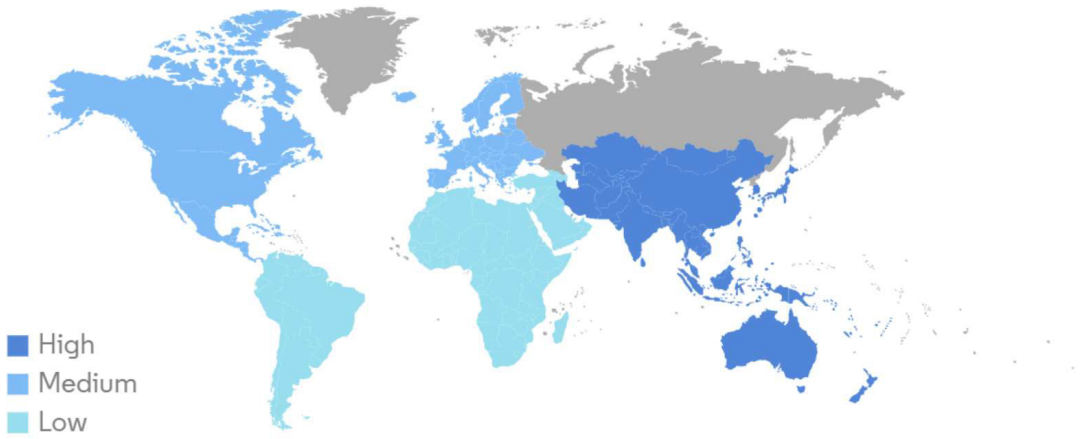


Figure 83: Assistive technologies for the blind market growth by region. Source: Mordor Intelligence (2022).

This industry is moderately consolidated, with some companies owning a good part of the market, such as Vispero, Humanware or Amedia Networks but it is still a competitive market. This allows the market to keep evolving and expanding, making incredible technological advancements that not only makes it profitable but also benefits society.

[49] Mordor Intelligence (retrieved in 2024).

5.2 COST ESTIMATION

In order to do a precise cost estimation, the first thing that needs to be understood is what exactly the selling of this product encompasses. In this project, the conceptual and software design of an ETA was created, and therefore those areas are the only ones inside the scope of costs. This means that the cost estimation will be focused on the expenses associated to these two areas. The main associated costs will be the technological components so that it can be tested, and the manpower needed for everything. In addition, labor costs for assembling the device can be considered which would also include software development and testing phases. The costs due to man hours will be divided into the following four areas: Concept development, hardware prototyping, software development, and integration and testing.

To start we will account for the cost of the components that form the ETA, since this is not only the easiest estimation but also the more precise.

Component	Estimated Cost in USD
Raspberry Pi 4 [50]	$ 35
Raspberry Pi Camera Module V2 [51]	$ 25
RP LiDAR A1 [52]	$ 85
HC – SR04 [53]	$ 3 * 4
Motor [54]	$ 50
Motor driver [55]	$ 10
Battery [56]	$ 50
Speaker/USB Sound Card/Microphone [57]	$ 10
Vibration Motor [58]	$ 1
GPS Module [59]	$ 20
Inertial Measurement Unit [60]	$ 15
TOTAL	$ 313

Figure 84: Estimated components cost. Source: Own elaboration (2024).

[50] Raspberry Pi (retrieved in 2024).
[51] Raspberry Pi (retrieved in 2024).
[52] Slamtec (retrieved in 2024).
[53] Cytron Technologies (retrieved in 2024).
[54] SparkFun (retrieved in 2024).
[55] SparkFun (retrieved in 2024).
[56] Digi-Key (retrieved in 2024).
[57] Adafruit (retrieved in 2024).
[58] SparkFun (retrieved in 2024).
[59] SparkFun (retrieved in 2024).
[60] SparkFun (retrieved in 2024).

The total component cost would be $ 313 with the current commercially set prices, this can of course vary depending on vendor, as well as the quantity bought since bulk orders usually come with discounts. It is also necessary to determine each of the component's lifecycle so that we know how often the ETA would need components to be repaired or substituted. In addition, the calculation of each of the parts amortization period will help spread out the cost and ensure a more manageable expense over time.

Component	Cost (USD)	Lifecycle (Years)	Amortization Period (Months)	Monthly Amortization (USD)
Raspberry Pi 4	$35	5	60	$0.58
Raspberry Pi Camera Module V2	$25	3	36	$0.69
RP LiDAR A1	$85	4	48	$1.77
HC-SR04 (4 units)	$12	2	24	$0.50
Motor	$50	4	48	$1.04
Motor Driver	$10	3	36	$0.28
Battery	$50	2	24	$2.08
Speaker/USB Sound Card/Microphone	$10	3	36	$0.28
Vibration Motor	$1	3	36	$0.03
GPS Module	$20	3	36	$0.56
IMU	$15	3	36	$0.42

Figure 85: Life cycle and amortization of components. Source: Own elaboration (2024)

For the manpower, each of the five sections will be analyzed, and based on project handbooks and information from other devices currently in the market an estimation will be done.

- Concept Development: This phase involves the initial design and its development. The estimated man- hours are 100 hours.

- Hardware Prototyping: This includes building and testing prototypes. The estimated man- hours are 200 hours.

- Software Development: This consists of writing, testing, and debugging the code that will run on the ETA. The estimated man- hours are 300 hours.

- Integration and Testing: In this phase both hardware and software are tested together to ensure everything works correctly. The estimated man-hours are 400 hours.

The total man hours needed for the scope of this project would be around 1000 hours. To then estimate the costs, we need to also estimate the hourly rates of the required labor, which will mainly be engineers. The engineers needed would be software engineers, hardware engineers, and systems engineers; in Spain they are paid around $ 19.43 [61], $ 22.48 [62] and $ 20.06 [63] respectively. We will assume that the concept development is done by systems engineers, the hardware prototyping by hardware engineers, the software development by software engineers and the integration and testing by systems engineers. With this information the man-hour associated costs can be calculated.

Area	Man-hours needed	Engineer needed	Engineer Salary	Associated Cost
Conceptual design	50 hours	Systems Engineer	$ 20.06 / hour	$ 1,003
Hardware prototyping	150 hours	Hardware Engineer	$ 22.48 / hour	$ 3,372
Software development	300 hours	Software Engineer	$ 19.43 / hour	$ 5,829
Integration and testing	400 hours	Systems Engineer	$ 20.06 / hour	$ 8,024
TOTAL COST				$ 18,228

Although it may seem high, this would only be spent once for the initial investment needed to further develop the ETA to a point where it is market ready.

In state of the art, two navigation devices with similar technologies were analyzed, the Glide and Lysa robots, therefore we can compare our projects cost with them. The Glide device is currently being sold at a price of $1,499 and Lysa has a price of $3,000. The Glide does not currently have their sales published but the owners of Lysa stated in 2023 that they expect to sell around 70,000 units by the end of 2026, meaning a total of 17,500 annually [64]. If our ETA were to follow the market sales of Lysa with the current calculated

[61] Jobted (retrieved in 2024).
[62] Average Salary Survey (retrieved in 2024).
[63] Salary (retrieved in 2024).
[64] MIT Solve (2023)

costs, each unit would cost almost nothing. Of course this isn't realistic since mass production, marketing, extra materials and many other costs were not taken into account since they fall out of the project's scope but due to the incredibly high margin for extra costs it can still be affirmed that it would be profitable.

5.3 CONCLUSIONS

In this chapter we have seen the extensive market of assistive technology for the visually impaired, currently worth around $ 13,2 billion. This is an expanding market, expected to keep growing at very high rates, due to advancements in technology and the increasing demand, mainly attributed to the aging of the population. Although a somewhat established industry there is still room for competitors, which allows for an ongoing evolution of this type of technology, with new innovative solutions being developed.

The cost analysis for this specific ETA was done by calculating the total component cost, as well as the needed man-hours for this projects scope. The initial, one-time cost was found to be around 18,228 dollars, which would be used to further test the robot by creating real prototypes that would allow to fix errors that were not fixed in the simulation. As for the component cost, this one would be a lot cheaper, around 313 dollars if the current market prices were used. Of course, this does not take into account the discounts due to bulk orders if this was to be industrialized and produces at a high scale. In addition, similar products were analyzed to see how it would compare and its possible profitability. Although the actual cost would be higher, it was found that the margin for an increased cost is extremely high and therefore even with the unaccounted costs this device could be very profitable as of now. Furthermore, with the market expansion these profits would only be expected to keep growing.

Chapter 6. CONCLUSION AND FUTURE WORKS

In the creation processes of this device several conclusions have been reached as to how to implement and design the perfect ETA, and different setbacks have had overcome. The purpose of this chapter is to highlight these conclusions and to show the future work that could be done to further improve.

6.1 RESULTS

The result of this project has been the creation of a new innovative solution designed to assist and guide the visually impaired when navigating an urban environment. In order to achieve a safe device that truly helps in every aspect of this navigation, the ETA has had to meet the objectives set in Chapter 1. Therefore, to make conclusions and study the product designed, whether or not those objectives were met and how efficiently will be analyzed. The primary goals outlined included effective obstacle detection, accurate navigation and geolocation, user feedback and portability.

Obstacle detection:

- Range of detection: The range has been successfully met for the y-axis, with the device being able to detect obstacle from ground level all the way up to head level. As for the x-axis a 360-degree range was able to be set, but only at waist level were the 2D LiDAR is positioned due to the reduced range of ultrasonic sensors. It was concluded that adding more ultrasonic sensors to the side or even to the back of the device would overcomplicate the system more than it would help user safety. This is due to the fact that side obstacle does not generate an imminent danger to the individual and in the odd case that they did the 2D LiDAR would still be able to detect almost all if not all of the obstacles.

- Static and dynamic obstacle: Static and dynamic obstacles are both evaded using the same avoidance logic which has allowed to reduce complexities even more. Although this avoidance logic has been tested for both static and moving obstacles and has worked an issue was encountered. Obstacles moving a higher speed than 10.98 km/h were not able to be tested since the human block used for the simulation lost stability when turning to try to avoid that obstacle. With that being said, the obstacle avoidance logic was still determined to be a success since the actual ETA did not destabilize and the turning logic still worked at higher speeds.

- Obstacle identification: This is divided in stair detection and traffic light detection. The main objective with this was to maintain the logic as simple as possible to one, not overcomplicate the device and to two, keep it affordable. For the stair detection the theory developed far exceeded expectations since no camera or machine learning has to be used. The simple algorithm developed with four ultrasonic sensors worked perfectly in the simulation and was able to detect up and down stairs from a safe distance. As for the traffic light a camera was indeed needed making the cost go up but the code behind was not complex. The detection was able to be done by just detecting colors and shapes. This was tested in the simulation to make sure only the color from the traffic lights were detected and it was a complete success with a return of zero errors. In addition, logic was implemented into the mapping so that the camera would only be turned on for short periods of time making it so that the battery life is longer.

- Range of distance detection: The objective here was to detect obstacles from a few centimeters all the way up to a few meters. The longer distance of detection was complete success with it being 25 meters for the camera, 6 meters for the LiDAR and 4 meters for the ultrasonic sensors. These ranges have proven to be more than enough so that the ETA can respond to those detections. As for the lower range of detection it was also achieved since all of the sensors chosen could detect obstacles from 10 to 2 cm away. Furthermore, another safety measure is the actual physical design since the first thing to come into contact with an obstacle be the ETA and not the user.

- Depth perception: This objective has not only been met but it has surpassed expectations since it was able to be implemented in a way that reduced complexity. It was done by implementing cluster logic, which treats obstacles near one another as one, reducing the number of points that need to analyze as well as reducing the complexity of the avoidance logic.

- Time processing: All of this was tested with a simulation where the response time was near to zero. This was thanks to the persisting objective of keeping things simple.

Geolocation and Navigation:

- Position: This was easily achieved with the use of a GPS.

- Self-orientation: Thanks to the implementation of an inertial measurement unit, the device is able to know the robot's yaw angle which tells the ETA the direction its pointing too.

- Route planning: The objective was to calculate the shortest route to go to the user's inputted desired destination. This was a complete success and done using the A* algorithm. With this algorithm the GPS and IMU information were also used, to truly make the path guiding as efficient as possible. This logic was tested and was found that it has a 100% success rate, always returning the optimal path. This was tested on low scale with less waypoints and edges than a real city would have but the testing gives us no reason to believe it wouldn't work on a bigger scale.

User Feedback:

- Detailed Feedback: The feedback to the visually impaired individual was determined to be via voice commands. This was tested in the simulation by printing out the comments that would be emitted verbally and was confirmed that each comment was emitted at the correct moment.

- Intuitive: It was very important for the comments to be precise and to the point which was exactly what was done. All of the feedback given was designed to be only one sentence and as to the point as possible.

- Real Time Feedback: This was again achieved and shown in the simulation with the feedback being almost instant.

Portability:

- Lightweight: In order to make as light as possible the least number of components was used, and the physical design was made to be extremely simple being only one wheel, and two bars. Although the exact weight is unknown since the materials have not been yet chosen it is expected to be very minimal which will allow the user to carry it with ease.

- Compact Size: This was taken into account for the physical design, and as stated in the previous paragraph this one was kept being very simple and therefore has a compact size. The size is comparable with a normal white cane, with the exception of the added wheel which allows the ETA to perfectly fit in any vehicle in order for it to be transported.

In conclusion all of the objectives have been successfully met to create an ETA capable of doing the following:

- Detect upward and downward staircases, as well as be able to go up and down them easily.
- Know the location of crosswalks and identify the traffic light color, to determine whether or not it is safe to cross.
- Determine the optimal path to the user's desired location and guiding them to it while staying on the sidewalk.
- Detect obstacles from ground level, all the way up to head level.
- Avoid obstacles when they are detected to be in the user's path.

All of this while remaining simple and as seen in Chapter 5 extremely affordable.

6.2 FUTURE WORKS

Technology as whole is in a constant state of improvement and innovation, in the last 10 years it has evolved more than anyone could imagine, with the development of technologies such as artificial intelligence and machine learning. This evolution has and will continue to have a great impact in the robotics industry, and therefore the possible future works for this device are practically infinite.

On a more present note, the device can always be improved with today's technologies. With time and investment more characteristics can be added to it, so that it is able to help the visually impaired even more than it already does, such as obstacle recognition for more objects or even face recognition. Obviously with this does come a higher cost and higher complexity, therefore it would be necessary to keep investigating on how to achieve these capabilities in a relatively simple way and cost-effective.

On the less electronical side of things and more mechanical, the next step would be to actually build this ETA and test it in real life. For this, the first step would be to further analyze the physical design, by studying the different materials that could be used and calculating the precise mechanical forces that it would be submitted to. In addition, a whole other study would have to be made in relation to the wheel, and the requisites it would need to meet to make a stable and most importantly safe robot.

Navigational aids not only for the visually impaired but as a whole are in continuous development and the number of possible future lines of work are immense. This industry is

gaining presence in today's world, and it is essential to keep innovating since it could help improve the lives of thousands of millions of people.

Chapter 7. BIBLIOGRAPHY

[1] MASAL, Komal Mahadeo; BHATLAWANDE, Shripad; SHINGADE, Sachin Dattatraya. Development of a visual to audio and tactile substitution system for mobility and orientation of visually impaired people: a review. Multimedia Tools and Applications, 2024, vol. 83, no 7, p. 20387-20427.

[2] American Community Survey (retrieved in 2024). Retrieved from: https://www.disabilitystatistics.org/

[3] Blasch, B.B.; Wiener, W.R.; Welsh, R.L. Foundations of Orientation and Mobility, 2nd ed.; AFB Press: New York, NY, USA, 1997.

[4] United Nations (retrieved in 2024). Sustainable Development Goals. Retrieved from: https://sdgs.un.org/goals

[5] International Guide Dog Federation (retrieved in 2024). History of Guide Dogs. Retrieved from: https://www.igdf.org.uk/guide-dogs/history-of-guide-dogs/

[6] STRONG, Philip. The history of the white cane. Tennessee Counc. Blind, 2009.

[7] DI MATTIA, Valentina, et al. Electromagnetic technology for a new class of electronic travel aids supporting the autonomous mobility of visually impaired people. Visually Impaired: Assistive Technologies, Challenges and Coping Strategies, 2016.

[8] ROMLAY, Muhammad Rabani Mohd, et al. Methodologies and evaluation of electronic travel aids for the visually impaired people: a review. Bulletin of Electrical Engineering and Informatics, 2021, vol. 10, no 3, p. 1747-1758.

[9] BankMyCell (retrieved in 2024). How many phones are in the world. Retrieved from: https://www.bankmycell.com/blog/how-many-phones-are-in-the-world

[10] THEODOROU, Paraskevi; TSILIGKOS, Kleomenis; MELIONES, Apostolos. Multi-Sensor Data Fusion Solutions for Blind and Visually Impaired: Research and Commercial Navigation Applications for Indoor and Outdoor Spaces. Sensors, 2023, vol. 23, no 12, p. 5411.

[11] TOA, Mubina; WHITEHEAD, Akeem. Ultrasonic sensing basics. Dallas: Texas Instruments, 2020, p. 53-75.

[12] DI MATTIA, Valentina, et al. Electromagnetic technology for a new class of electronic travel aids supporting the autonomous mobility of visually impaired people. Visually Impaired: Assistive Technologies, Challenges and Coping Strategies, 2016.

[13] MOCANU, Bogdan; TAPU, Ruxandra; ZAHARIA, Titus. When ultrasonic sensors and computer vision join forces for efficient obstacle detection and recognition. Sensors, 2016, vol. 16, no 11, p. 1807.

[14] MORGAN, Elijah J. HC-SR04 ultrasonic sensor. Nov, 2014.

[15] AJMERA, Pooja. A review paper on infrared sensor. International Journal of Engineering Research & Technology (IJERT), 2017, vol. 5, no 23, p. 1-3.

[16] RASHID, A.; ALI, A. Performance analysis of low-cost infrared sensors for multi-robot localization and communication. Int. J. Comput. Appl, 2018, vol. 182, p. 23-29.

[17] Sharp Corporation. GP2Y0A02YK0F Long Distance Measuring Sensor. 2006. Retrieved from: https://global.sharp/products/device/lineup/data/pdf/datasheet/gp2y0a02yk_e.pdf

[18] Sharp Corporation. GP2Y0A710K0F Long Distance Measuring Sensor. 2006. Retrieved from: https://global.sharp/products/device/lineup/data/pdf/datasheet/gp2y0a710k_e.pdf

[19] LI, You; IBANEZ-GUZMAN, Javier. Lidar for autonomous driving: The principles, challenges, and trends for automotive lidar and perception systems. IEEE Signal Processing Magazine, 2020, vol. 37, no 4, p. 50-61.

[20] BEHROOZPOUR, Behnam, et al. Lidar system architectures and circuits. IEEE Communications Magazine, 2017, vol. 55, no 10, p. 135-142.

[21] SLAMTEC. RPLIDAR A1M8 360 Degree Laser Scanner Development Kit Datasheet. 2017. Retrieved from: https://bucket-download.slamtec.com/d1e428e7efbdcd65a8ea111061794fb8d4ccd3a0/LD108_SLAMTEC_rplidar_datasheet_A1M8_v3.0_en.pdf

[22] MUKHIDDINOV, Mukhriddin; CHO, Jinsoo. Smart glass system using deep learning for the blind and visually impaired. Electronics, 2021, vol. 10, no 22, p. 2756.

[23] DHOD, Rahul, et al. Low-cost GPS and GSM based navigational aid for visually impaired people. Wireless Personal Communications, 2017, vol. 92, p. 1575-1589.

[24] AHMAD, Norhafizan, et al. Reviews on various inertial measurement unit (IMU) sensor applications. International Journal of Signal Processing Systems, 2013, vol. 1, no 2, p. 256-262.

[25] ANDREJAŠIC, Matej. Mems accelerometers. En University of Ljubljana. Faculty for mathematics and physics, Department of physics, Seminar. 2008.

[26] PASSARO, Vittorio MN, et al. Gyroscope technology and applications: A review in the industrial perspective. Sensors, 2017, vol. 17, no 10, p. 2284.

[27] LI, Wei; WANG, Jinling. Magnetic sensors for navigation applications: an overview. The Journal of navigation, 2014, vol. 67, no 2, p. 263-275.

[28] Código Técnico de la Edificación. Documento Básico de Seguridad y Accesibilidad. Retreived from: https://www.codigotecnico.org/pdf/Documentos/SUA/DBSUA.pdf.

[29] BOUHAMED, Sonda Ammar; KALLEL, Imene Khanfir; MASMOUDI, Dorra Sellami. New electronic white cane for stair case detection and recognition using ultrasonic sensor. International Journal of Advanced Computer Science and Applications, 2013, vol. 4, no 6.

[30] Pepperl+Fuchs. (2019). Ultrasonic Sensor FAQ: Synchronization and Common Mode. Retreived from: https://blog.pepperl-fuchs.com/en/2019/ultrasonic-sensor-faq-synchronization-and-common-mode/.

[31] GHORPADE, Deepali; THAKARE, Anuradha D.; DOIPHODE, Sunil. Obstacle detection and avoidance algorithm for autonomous mobile robot using 2D LiDAR. En 2017 International

Conference on Computing, Communication, Control and Automation (ICCUBEA). IEEE, 2017. p. 1-6.

[32] MathWorks (retrieved in 2024). ROI-Based Processing. Retrieved from: https://es.mathworks.com/help/images/roi-based-processing.html

[33] WONGHABUT, Pasit, et al. Traffic light color identification for automatic traffic light violation detection system. En 2018 international conference on engineering, applied sciences, and technology (ICEAST). IEEE, 2018. p. 1-4.

[34] HASSAN, Nazirah; MING, Kong Wai; WAH, Choo Keng. A comparative study on hsv-based and deep learning-based object detection algorithms for pedestrian traffic light signal recognition. En 2020 3rd International Conference on Intelligent Autonomous Systems (ICoIAS). IEEE, 2020. p. 71-76.

[35] NAYYAR, Anand; PURI, Vikram. Raspberry Pi-a small, powerful, cost effective and efficient form factor computer: a review. International Journal of Advanced Research in Computer Science and Software Engineering, 2015, vol. 5, no 12, p. 720-737. [36] Raspberry Pi (retrieved in 2024). Raspberry Pi Camera Module V2 Documentation. Retrieved from: https://www.raspberrypi.org/documentation/accessories/camera.html

[37] Slamtec (retrieved in 2024). RPLIDAR A1 Documentation. Retrieved from: https://www.slamtec.com/en/Lidar/A1

[38] Cytron Technologies (2023). "HC-SR04 Ultrasonic Sensor." Retrieved from Cytron Technologies : https://www.cytron.io/p-5v-hc-sr04-ultrasonic-sensor

[39] SparkFun (2023). "DC Motor with Gearbox." Retrieved from: https://dronebotworkshop.com/dc-motor-drivers/

[40] Raspberry Pi (retrieved in 2024). Using USB Audio with Raspberry Pi. Retrieved from: https://www.raspberrypi.org/documentation/computers/configuration.html#usb-audio

[41] Raspberry Pi Forums (retrieved in 2024). USB Sound Cards for Raspberry Pi. Retrieved from: https://www.raspberrypi.org/forums/viewtopic.php?t=8496

[42] Pololu (retrieved in 2024). Using Vibration Motors with Arduino and Raspberry Pi. Retrieved from: https://www.pololu.com/product/1637

[43] Adafruit (retrieved in 2024). Adafruit Ultimate GPS. Retrieved from: https://learn.adafruit.com/adafruit-ultimate-gps

[44] Adafruit (retrieved in 2024). MPU6050 6-DOF Accelerometer and Gyroscope. Retrieved from: https://learn.adafruit.com/mpu6050-6-dof-accelerometer-and-gyroscope

[45] Math.oxford.emory.edu (retrieved in 2024). A Algorithm. Retrieved from: https://math.oxford.emory.edu/site/cs171/theAStarAlgorithm/*

[46] Cs.cornell.edu. (2007). Recitation 26: Heuristics and A. Retrieved from: https://www.cs.cornell.edu/courses/cs312/2007sp/recitations/rec26.html*

[47] Theory.stanford.edu (retrieved in 2024). Heuristics for A and Other Search Algorithms. Retrieved from: https://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html*

[48] Verified Market Reports (retrieved in 2024). Assistive Technologies for Visually Impaired Market. Retrieved from: https://www.verifiedmarketreports.com/product/assistive-technologies-for-visually-impaired-market/#:~:text=Assistive%20Technologies%20for%20Visually%20Impaired%20Market%20size%20was%20valued%20at,the%20forecast%20period%202024%2D2030

[49] Mordor Intelligence (retrieved in 2024). Assistive Technologies for Visually Impaired Market. Retrieved from: https://www.mordorintelligence.com/industry-reports/assistive-technologies-for-visually-impaired-market

[50] Raspberry Pi (retrieved in 2024). "Buy a Raspberry Pi 4 Model B." Retrieved from: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

[51] Seeed Studio (retrieved in 2024). "Raspberry Pi Camera Module V2." Retrieved from: https://www.raspberrypi.com/products/camera-module-v2/

[52] Slamtec (retrieved in 2024). "RPLIDAR A1 360 Degree Laser Scanner Development Kit." Retrieved from: https://www.slamtec.ai/product/slamtec-rplidar-a1/

[53] Cytron Technologies (retrieved in 2024). "HC-SR04 Ultrasonic Sensor." Retrieved from: https://www.cytron.io/p-5v-hc-sr04-ultrasonic-sensor

[54] SparkFun (retrieved in 2024). "DC Motor with Gearbox." Retrieved from: https://dronebotworkshop.com/dc-motor-drivers/

[55] SparkFun (retrieved in 2024). "L298N Dual H-Bridge Motor Driver." Retrieved from: https://robu.in/product-category/dc-motors/motor-drivers/brushed-dc-motor-driver/

[56] Digi-Key (retrieved in 2024). "Rechargeable Battery Packs." Retrieved from: https://www.digikey.com/en/articles/selecting-batteries-for-iot-devices

[57] Adafruit (retrieved in 2024). "Mini Metal Speaker." Retrieved from: https://www.adafruit.com/product/1890

[58] SparkFun (retrieved in 2024). "Vibration Motor." Retrieved from: https://uptowncraftworks.com/how-much-does-a-hobby-vibration-motor-cost/

[59] SparkFun (retrieved in 2024). "GPS Module." Retrieved from: https://www.sparkfun.com/products/19166

[60] SparkFun (retrieved in 2024). "MPU-9250 9-DOF IMU Breakout." Retrieved from: https://www.sparkfun.com/products/retired/13762

[61] Jobted (retrieved in 2024). Salario Ingeniero Software. Retrieved from: https://www.jobted.es/salario/ingeniero-software

[62] Average Salary Survey (retrieved in 2024). Hardware Engineer Salary in Spain. Retrieved from: https://www.averagesalarysurvey.com/hardware-engineer/spain

[63] Salary.com (retrieved in 2024). Systems Engineer I Salary in Spain. Retrieved from: https://www.salary.com/research/es-salary/benchmark/systems-engineer-i-salary/es

[64] MIT Solve (2023). "Heath in Fragile Contexts Challenge: Solution #70942." Retrieved from: https://solve.mit.edu/challenges/heath-in-fragile-contexts-challenge/solutions/70942

[65] BEINGOLEA, Jorge Rodolfo, et al. Assistive devices: technology development for the visually impaired. Designs, 2021, vol. 5, no 4, p. 75.

ANNEX I: CODE

DEVICE CONTROLLER

```
import numpy as np
from controller import Camera
from controller import DistanceSensor
from controller import Robot, GPS, InertialUnit, Emitter, Lidar
from math import sqrt, atan2, pi, cos, sin
import networkx as nx
import heapq
import struct
import time
import math
import cv2

# Create the Robot instance
robot = Robot()

timestep = int(robot.getBasicTimeStep())

# Initialize sensors
s1 = robot.getDevice('s1')
s1.enable(timestep)
s2 = robot.getDevice('s2')
s2.enable(timestep)
s3 = robot.getDevice('s3')
s3.enable(timestep)
s4 = robot.getDevice('s4')
s4.enable(timestep)

camera = robot.getDevice('camera')
camera.enable(timestep)

gps = robot.getDevice('gps')
gps.enable(timestep)

imu = robot.getDevice('inertial unit')
imu.enable(timestep)

emitter = robot.getDevice('emitter')

lidar = robot.getDevice('lidar')
lidar.enable(timestep)
lidar.enablePointCloud()
```

```
# Set the goal coordinates (these should match one of the waypoints)
goal_x = 33.3
goal_y = -43.9

# Define movement commands
stop = 0
forward = 1

# Constants
speed_of_sound = 343.0  # m/s
max_distance = 4.0  # meters
d_S1_w = 0.48416
d_S2_w = 0.53114
d_S3_w = 0.57813
velocity = 11.1

# Calculate the maximum time for a round trip for the ultrasonic wave.
T_i = (max_distance * 4) / speed_of_sound

counter_up = 0
last_value_up = False
counter_down = 0
last_value_down = False

# Traffic light detection
def detect_traffic_light(image):
    # Convert the image to a format that OpenCV can process
    np_image = np.frombuffer(image, dtype=np.uint8).reshape((camera.getHeight(),
camera.getWidth(), 4))
    np_image = cv2.cvtColor(np_image, cv2.COLOR_RGBA2BGR)  # Convert RGBA to
BGR for OpenCV

    # Define the region of interest (ROI) where the traffic light is expected to appear
    height, width, _ = np_image.shape
    roi_top = 0  # Start from the top
    roi_bottom = height // 2  # Cover the top half
    roi_left = 0  # Cover the full width
    roi_right = width
    roi = np_image[roi_top:roi_bottom, roi_left:roi_right]

    # Convert the ROI to HSV color space
    hsv_image = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)

    # Define color ranges for red and green
    red_lower = np.array([120, 100, 110])
    red_upper = np.array([179, 255, 255])
    green_lower = np.array([35, 100, 100])
    green_upper = np.array([95, 255, 255])

    red_mask = cv2.inRange(hsv_image, red_lower, red_upper)
```

```
    green_mask = cv2.inRange(hsv_image, green_lower, green_upper)

    # Apply median blur to reduce noise
    red_mask = cv2.medianBlur(red_mask, 5)
    green_mask = cv2.medianBlur(green_mask, 5)

    # Apply morphological operations to clean up the mask
    kernel = np.ones((5, 5), np.uint8)
    red_mask = cv2.morphologyEx(red_mask, cv2.MORPH_CLOSE, kernel)
    green_mask = cv2.morphologyEx(green_mask, cv2.MORPH_CLOSE, kernel)

    # Function to detect circular shapes
    def detect_circles(mask):
        circles = cv2.HoughCircles(mask, cv2.HOUGH_GRADIENT, dp=1.2, minDist=50,
                                   param1=50, param2=30, minRadius=10, maxRadius=50)
        return circles is not None

    # Function to detect human-shaped contours
    def detect_human_shape(contours):
        for contour in contours:
            area = cv2.contourArea(contour)
            if area > 100:  # Filter out small areas
                perimeter = cv2.arcLength(contour, True)
                approx = cv2.approxPolyDP(contour, 0.04 * perimeter, True)
                if len(approx) > 5:  # Assuming human shape has more than 5 edges
                    return True
        return False

    # Function to filter out unwanted contours based on aspect ratio and area
    def filter_contours(contours):
        filtered_contours = []
        for contour in contours:
            area = cv2.contourArea(contour)
            x, y, w, h = cv2.boundingRect(contour)
            aspect_ratio = float(w) / h
            if 0.5 < aspect_ratio < 1.5 and area > 100:  # Aspect ratio close to 1 and
reasonable area
                filtered_contours.append(contour)
        return filtered_contours

    # Find and filter contours for red and green masks
    red_contours, _ = cv2.findContours(red_mask, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
    green_contours, _ = cv2.findContours(green_mask, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)

    red_contours = filter_contours(red_contours)
    green_contours = filter_contours(green_contours)

    # Check for circles and human shapes in both red and green masks
```

```
    red_circle_detected = detect_circles(red_mask)
    green_circle_detected = detect_circles(green_mask)
    red_human_detected = detect_human_shape(red_contours)
    green_human_detected = detect_human_shape(green_contours)

    # Determine the size and position of the detected objects
    red_count = sum(cv2.contourArea(c) for c in red_contours) if red_circle_detected or
red_human_detected else 0
    green_count = sum(cv2.contourArea(c) for c in green_contours) if
green_circle_detected or green_human_detected else 0

    position = gps.getValues()
    x_pos = 188 + position[0] + 9
    x_pos_rounded = round(x_pos)
    red_count_rounded = round(red_count)
    green_count_rounded = round(green_count)
    # Debugging: print the pixel counts
    print(f"{red_count_rounded}")

    # Define thresholds for detecting colors
    threshold = 720  # Adjust this threshold as needed

    if red_count > threshold:
        return 1  # Red light
    elif green_count > threshold:
        return 3  # Green light
    else:
        return 0  # None

last_traffic_light_state = 0  # Initialize last detected state
persistence_counter_red = 0
persistence_counter_green = 0
persistence_limit = 5  # Number of loops to persist the state

#Stair case detection
def calculate_horizontal_distance(sensor_value, angle_rad):
    return sensor_value * math.cos(angle_rad)

def calculate_distance_to_start_of_staircase(d_i_h, d_Si_w, d_i_s=5):
    return d_i_h - d_Si_w - d_i_s - 0.05

def detect_stairs(s1_value, s2_value, s3_value):

    upward_staircase_allowed_false = 40  # Number of allowed false readings within the
threshold
    downward_staircase_allowed_false = 40  # Number of allowed false readings within the
threshold
    global counter_up, last_value_up, counter_down, last_value_down

    # Define horizontal angles for each sensor in degrees
```

```
    h_angle_s1 = 24.75
    h_angle_s2 = 34
    h_angle_s3 = 48

    # Define constant vertical distance of the sensors
    s1_constant = 1.014118
    s2_constant = 0.9917609
    s3_constant = 0.9724122

    # Convert angles to radians
    h_angle_s1_rad = math.radians(h_angle_s1)
    h_angle_s2_rad = math.radians(h_angle_s2)
    h_angle_s3_rad = math.radians(h_angle_s3)

    # Calculate the vertical distances
    s1_vertical_distance = s1_value * math.sin(h_angle_s1_rad)
    s2_vertical_distance = s2_value * math.sin(h_angle_s2_rad)
    s3_vertical_distance = s3_value * math.sin(h_angle_s3_rad)


    # 1
    diff_s1 = (s1_constant - s1_vertical_distance)
    diff_s2 = (s2_constant - s2_vertical_distance)
    diff_s3 = (s3_constant - s3_vertical_distance)

    # print(f"Rise detected s1: {diff_s1}")

    # Check for upward staircase detection
    if 0.14 <= diff_s1 <= 0.17:
        upward_staircase_detected = True
        d_1_s = 0.1 + 0.16*0
        d_i_h = calculate_horizontal_distance(s1_value, h_angle_s1_rad)
        distance_to_start = calculate_distance_to_start_of_staircase(d_i_h, d_S1_w, d_1_s)
        last_value_up = upward_staircase_detected
        counter_up = 0
        if distance_to_start >= 1:
            print(f"Upward staircase in: {distance_to_start} meters")
        if distance_to_start <= 0.01:
            print(f"In contact with upward staircase")

    else:
        # Check N = 2
        diff_s1_2 = (s1_constant - s1_vertical_distance) / 2
        if 0.14 <= diff_s1_2 <= 0.17:
            upward_staircase_detected = True
            d_1_s = 0.1 + 0.30*1
            d_i_h = calculate_horizontal_distance(s1_value, h_angle_s1_rad)
            distance_to_start = calculate_distance_to_start_of_staircase(d_i_h, d_S1_w,
d_1_s)
            last_value_up = upward_staircase_detected
```

```
            counter_up = 0
            if distance_to_start >= 1:
                print(f"Upward staircase in: {distance_to_start} meters")
            if distance_to_start <= 0.01:
                print(f"In contact with upward staircase")
        else:
            # Check N = 3
            diff_s1_3 = (s1_constant - s1_vertical_distance) / 3
            if 0.14 <= diff_s1_3 <= 0.17:
                upward_staircase_detected = True
                d_1_s = 0.1 + 0.30*2
                d_i_h = calculate_horizontal_distance(s1_value, h_angle_s1_rad)
                distance_to_start = calculate_distance_to_start_of_staircase(d_i_h, d_S1_w,
d_1_s)
                last_value_up = upward_staircase_detected
                counter_up = 0
                if distance_to_start >= 1:
                    print(f"Upward staircase in: {distance_to_start} meters")
                if distance_to_start <= 0.01:
                    print(f"In contact with upward staircase")
            else:
                # Check N = 4
                diff_s1_4 = (s1_constant - s1_vertical_distance) / 4
                if 0.14 <= diff_s1_4 <= 0.17:
                    upward_staircase_detected = True
                    d_1_s = 0.1 + 0.30*3
                    d_i_h = calculate_horizontal_distance(s1_value, h_angle_s1_rad)
                    distance_to_start = calculate_distance_to_start_of_staircase(d_i_h, d_S1_w,
d_1_s)
                    last_value_up = upward_staircase_detected
                    counter_up = 0
                    if distance_to_start >= 1:
                        print(f"Upward staircase in: {distance_to_start} meters")
                    if distance_to_start <= 0.01:
                        print(f"In contact with upward staircase")
                else:
                    # Check N = 5
                    diff_s1_5 = (s1_constant - s1_vertical_distance) / 5
                    if 0.14 <= diff_s1_5 <= 0.17:
                        upward_staircase_detected = True
                        d_1_s = 0.1 + 0.30*4
                        d_i_h = calculate_horizontal_distance(s1_value, h_angle_s1_rad)
                        distance_to_start = calculate_distance_to_start_of_staircase(d_i_h,
d_S1_w, d_1_s)
                        last_value_up = upward_staircase_detected
                        counter_up = 0
                        if distance_to_start >= 1:
                            print(f"Upward staircase in: {distance_to_start} meters")
                        if distance_to_start <= 0.01:
                            print(f"In contact with upward staircase")
```

```
            else:
                # Check N = 6
                diff_s1_6 = (s1_constant - s1_vertical_distance) / 6
                if 0.14 <= diff_s1_6 <= 0.17:
                    upward_staircase_detected = True
                    d_1_s = 0.1 + 0.30*5
                    d_i_h = calculate_horizontal_distance(s1_value, h_angle_s1_rad)
                    distance_to_start = calculate_distance_to_start_of_staircase(d_i_h,
d_S1_w, d_1_s)
                    last_value_up = upward_staircase_detected
                    counter_up = 0
                    if distance_to_start >= 1:
                        print(f"Upward staircase in: {distance_to_start} meters")
                    if distance_to_start <= 0.01:
                        print(f"In contact with upward staircase")
                else:
                    # Check N = 7
                    diff_s1_7 = (s1_constant - s1_vertical_distance) / 7
                    if 0.14 <= diff_s1_7 <= 0.17:
                        upward_staircase_detected = True
                        d_1_s = 0.1 + 0.30*6
                        d_i_h = calculate_horizontal_distance(s1_value, h_angle_s1_rad)
                        distance_to_start = calculate_distance_to_start_of_staircase(d_i_h,
d_S1_w, d_1_s)
                        last_value_up = upward_staircase_detected
                        counter_up = 0
                        if distance_to_start >= 1:
                            print(f"Upward staircase in: {distance_to_start} meters")
                        if distance_to_start <= 0.01:
                            print(f"In contact with upward staircase")
                    else:
                        # Check N = 8
                        diff_s1_8 = (s1_constant - s1_vertical_distance) / 8
                        if 0.14 <= diff_s1_8 <= 0.17:
                            upward_staircase_detected = True
                            d_1_s = 0.1 + 0.30*7
                            d_i_h = calculate_horizontal_distance(s1_value, h_angle_s1_rad)
                            distance_to_start = calculate_distance_to_start_of_staircase(d_i_h,
d_S1_w, d_1_s)
                            last_value_up = upward_staircase_detected
                            counter_up = 0
                            if distance_to_start >= 1:
                                print(f"Upward staircase in: {distance_to_start} meters")
                            if distance_to_start <= 0.01:
                                print(f"In contact with upward staircase")
                        else:
                            # Check N = 9
                            diff_s1_9 = (s1_constant - s1_vertical_distance) / 9
                            if 0.14 <= diff_s1_9 <= 0.17:
                                upward_staircase_detected = True
```

```
                            d_1_s = 0.1 + 0.30*8
                            d_i_h = calculate_horizontal_distance(s1_value,
h_angle_s1_rad)
                            distance_to_start =
calculate_distance_to_start_of_staircase(d_i_h, d_S1_w, d_1_s)
                            last_value_up = upward_staircase_detected
                            counter_up = 0
                            if distance_to_start >= 1:
                                print(f"Upward staircase in: {distance_to_start} meters")
                            if distance_to_start <= 0.01:
                                print(f"In contact with upward staircase")
                        else:
                            # Check N = 10
                            diff_s1_10 = (s1_constant - s1_vertical_distance) / 10
                            if 0.14 <= diff_s1_10 <= 0.17:
                                upward_staircase_detected = True
                                d_1_s = 0.1 + 0.30*9
                                d_i_h = calculate_horizontal_distance(s1_value,
h_angle_s1_rad)
                                distance_to_start =
calculate_distance_to_start_of_staircase(d_i_h, d_S1_w, d_1_s)
                                last_value_up = upward_staircase_detected
                                counter_up = 0
                                if distance_to_start >= 1:
                                    print(f"Upward staircase in: {distance_to_start} meters")
                                if distance_to_start <= 0.01:
                                    print(f"In contact with upward staircase")
                            else:
                                upward_staircase_detected = last_value_up
                                counter_up += 1
                                if counter_up > upward_staircase_allowed_false:
                                    upward_staircase_detected = False

    # Check for downward staircase detection s2
    if -0.17 <= diff_s2 <= -0.14:
        downward_staircase_detected2 = True
        d_2_s = 0.24614 + 0.30*0
        d_i_h = calculate_horizontal_distance(s2_value, h_angle_s2_rad)
        distance_to_start = calculate_distance_to_start_of_staircase(d_i_h, d_S2_w, d_2_s)
        last_value_down = downward_staircase_detected2
        counter_down = 0
        if distance_to_start >= 0.85:
            print(f"Downward staircase in: {distance_to_start} meters")
        if distance_to_start <= 0.01:
            print(f"In contact with downward staircase")
    else:
        # Check N = 2
        diff_s2_2 = (s2_constant - s2_vertical_distance) / 2
        if -0.17 <= diff_s2_2 <= -0.14:
            downward_staircase_detected2 = True
```

```
        d_2_s = 0.24614 + 0.30*1
        d_i_h = calculate_horizontal_distance(s2_value, h_angle_s2_rad)
        distance_to_start = calculate_distance_to_start_of_staircase(d_i_h, d_S2_w,
d_2_s)
        last_value_down = downward_staircase_detected2
        counter_down = 0
        if distance_to_start >= 0.85:
            print(f"Downward staircase in: {distance_to_start} meters")
        if distance_to_start <= 0.01:
            print(f"In contact with downward staircase")
    else:
        # Check N = 3
        diff_s2_3 = (s2_constant - s2_vertical_distance) / 3
        if -0.17 <= diff_s2_3 <= -0.14:
            downward_staircase_detected2 = True
            d_2_s = 0.24614 + 0.30*2
            d_i_h = calculate_horizontal_distance(s2_value, h_angle_s2_rad)
            distance_to_start = calculate_distance_to_start_of_staircase(d_i_h, d_S2_w,
d_2_s)
            last_value_down = downward_staircase_detected2
            counter_down = 0
            if distance_to_start >= 0.85:
                print(f"Downward staircase in: {distance_to_start} meters")
            if distance_to_start <= 0.01:
                print(f"In contact with downward staircase")
        else:
            # Check N = 4
            diff_s2_4 = (s2_constant - s2_vertical_distance) / 4
            if -0.17 <= diff_s2_4 <= -0.14:
                downward_staircase_detected2 = True
                d_2_s = 0.24614 + 0.30*3
                d_i_h = calculate_horizontal_distance(s2_value, h_angle_s2_rad)
                distance_to_start = calculate_distance_to_start_of_staircase(d_i_h, d_S2_w,
d_2_s)
                last_value_down = downward_staircase_detected2
                counter_down = 0
                if distance_to_start >= 0.85:
                    print(f"Downward staircase in: {distance_to_start} meters")
                if distance_to_start <= 0.01:
                    print(f"In contact with downward staircase")
            else:
                # Check N = 5
                diff_s2_5 = (s2_constant - s2_vertical_distance) / 5
                if -0.17 <= diff_s2_5 <= -0.14:
                    downward_staircase_detected2 = True
                    d_2_s = 0.24614 + 0.30*4
                    d_i_h = calculate_horizontal_distance(s2_value, h_angle_s2_rad)
                    distance_to_start = calculate_distance_to_start_of_staircase(d_i_h,
d_S2_w, d_2_s)
                    last_value_down = downward_staircase_detected2
```

```
                counter_down = 0
                if distance_to_start >= 0.85:
                    print(f"Downward staircase in: {distance_to_start} meters")
                if distance_to_start <= 0.01:
                    print(f"In contact with downward staircase")
            else:
                # Check N = 6
                diff_s2_6 = (s2_constant - s2_vertical_distance) / 6
                if -0.17 <= diff_s2_6 <= -0.14:
                    downward_staircase_detected2 = True
                    d_2_s = 0.24614 + 0.30*5
                    d_i_h = calculate_horizontal_distance(s2_value, h_angle_s2_rad)
                    distance_to_start = calculate_distance_to_start_of_staircase(d_i_h,
d_S2_w, d_2_s)
                    last_value_down = downward_staircase_detected2
                    counter_down = 0
                    if distance_to_start >= 0.85:
                        print(f"Downward staircase in: {distance_to_start} meters")
                    if distance_to_start <= 0.01:
                        print(f"In contact with downward staircase")
                else:
                    # Check N = 7
                    diff_s2_7 = (s2_constant - s2_vertical_distance) / 7
                    if -0.17 <= diff_s2_7 <= -0.14:
                        downward_staircase_detected2 = True
                        d_2_s = 0.24614 + 0.3*6
                        d_i_h = calculate_horizontal_distance(s2_value, h_angle_s2_rad)
                        distance_to_start = calculate_distance_to_start_of_staircase(d_i_h,
d_S2_w, d_2_s)
                        last_value_down = downward_staircase_detected2
                        counter_down = 0
                        if distance_to_start >= 0.85:
                            print(f"Downward staircase in: {distance_to_start} meters")
                        if distance_to_start <= 0.01:
                            print(f"In contact with downward staircase")
                    else:
                        # Check N = 8
                        diff_s2_8 = (s2_constant - s2_vertical_distance) / 8
                        if -0.17 <= diff_s2_8 <= -0.14:
                            downward_staircase_detected2 = True
                            d_2_s = 0.24614 + 0.3*7
                            d_i_h = calculate_horizontal_distance(s2_value, h_angle_s2_rad)
                            distance_to_start = calculate_distance_to_start_of_staircase(d_i_h,
d_S2_w, d_2_s)
                            last_value_down = downward_staircase_detected2
                            counter_down = 0
                            if distance_to_start >= 0.85:
                                print(f"Downward staircase in: {distance_to_start} meters")
                            if distance_to_start <= 0.01:
                                print(f"In contact with downward staircase")
```

```
                    else:
                        # Check N = 9
                        diff_s2_9 = (s2_constant - s2_vertical_distance) / 9
                        if -0.17 <= diff_s2_9 <= -0.14:
                            downward_staircase_detected2 = True
                            d_2_s = 0.24614 + 0.3*8
                            d_i_h = calculate_horizontal_distance(s2_value,
h_angle_s2_rad)
                            distance_to_start =
calculate_distance_to_start_of_staircase(d_i_h, d_S2_w, d_2_s)
                            last_value_down = downward_staircase_detected2
                            counter_down = 0
                            if distance_to_start >= 0.85:
                                print(f"Downward staircase in: {distance_to_start} meters")
                            if distance_to_start <= 0.01:
                                print(f"In contact with downward staircase")
                        else:
                            # Check N = 10
                            diff_s2_10 = (s2_constant - s2_vertical_distance) / 10
                            if -0.17 <= diff_s2_10 <= -0.14:
                                downward_staircase_detected2 = True
                                d_2_s = 0.24614 + 0.3*9
                                d_i_h = calculate_horizontal_distance(s2_value,
h_angle_s2_rad)
                                distance_to_start =
calculate_distance_to_start_of_staircase(d_i_h, d_S2_w, d_2_s)
                                last_value_down = downward_staircase_detected2
                                counter_down = 0
                                if distance_to_start >= 0.85:
                                    print(f"Downward staircase in: {distance_to_start} meters")
                                if distance_to_start <= 0.01:
                                    print(f"In contact with downward staircase")
                            else:
                                downward_staircase_detected2 = last_value_down
                                counter_down += 1
                                if counter_down > downward_staircase_allowed_false:
                                    downward_staircase_detected2 = False

    # Check for downward staircase detection s3
    if -0.17 <= diff_s3 <= -0.14:
        downward_staircase_detected3 = True
        d_3_s = 0.21017 + 0.30*0
        d_i_h = calculate_horizontal_distance(s3_value, h_angle_s3_rad)
        distance_to_start = calculate_distance_to_start_of_staircase(d_i_h, d_S3_w, d_3_s)
        last_value_down = downward_staircase_detected3
        counter_down = 0
        if distance_to_start >= 1:
            print(f"Downward staircase in: {distance_to_start} meters")
        if distance_to_start <= 0.01:
            print(f"In contact with downward staircase")
```

```
    else:
        # Check N = 2
        diff_s3_2 = (s3_constant - s3_vertical_distance) / 2
        if -0.17 <= diff_s3_2 <= -0.14:
            downward_staircase_detected3 = True
            d_3_s = 0.21017 + 0.3*1
            d_i_h = calculate_horizontal_distance(s3_value, h_angle_s3_rad)
            distance_to_start = calculate_distance_to_start_of_staircase(d_i_h, d_S3_w,
d_3_s)
            last_value_down = downward_staircase_detected3
            counter_down = 0
            if distance_to_start >= 1:
                print(f"Downward staircase in: {distance_to_start} meters")
            if distance_to_start <= 0.01:
                print(f"In contact with downward staircase")
        else:
            # Check N = 3
            diff_s3_3 = (s3_constant - s3_vertical_distance) / 3
            if -0.17 <= diff_s3_3 <= -0.14:
                downward_staircase_detected3 = True
                d_3_s = 0.21017 + 0.3*2
                d_i_h = calculate_horizontal_distance(s3_value, h_angle_s3_rad)
                distance_to_start = calculate_distance_to_start_of_staircase(d_i_h, d_S3_w,
d_3_s)
                last_value_down = downward_staircase_detected3
                counter_down = 0
                if distance_to_start >= 1:
                    print(f"Downward staircase in: {distance_to_start} meters")
                if distance_to_start <= 0.01:
                    print(f"In contact with downward staircase")
            else:
                # Check N = 4
                diff_s3_4 = (s3_constant - s3_vertical_distance) / 4
                if -0.17 <= diff_s3_4 <= -0.14:
                    downward_staircase_detected3 = True
                    d_3_s = 0.2 + 0.3*3
                    d_i_h = calculate_horizontal_distance(s3_value, h_angle_s3_rad)
                    distance_to_start = calculate_distance_to_start_of_staircase(d_i_h, d_S3_w,
d_3_s)
                    last_value_down = downward_staircase_detected3
                    counter_down = 0
                    if distance_to_start >= 1:
                        print(f"Downward staircase in: {distance_to_start} meters")
                    if distance_to_start <= 0.01:
                        print(f"In contact with downward staircase")
                else:
                    # Check N = 5
                    diff_s3_5 = (s3_constant - s3_vertical_distance) / 5
                    if -0.17 <= diff_s3_5 <= -0.14:
                        downward_staircase_detected3 = True
```

```
                d_3_s = 0.21017 + 0.3*4
                d_i_h = calculate_horizontal_distance(s3_value, h_angle_s3_rad)
                distance_to_start = calculate_distance_to_start_of_staircase(d_i_h,
d_S3_w, d_3_s)
                last_value_down = downward_staircase_detected3
                counter_down = 0
                if distance_to_start >= 1:
                    print(f"Downward staircase in: {distance_to_start} meters")
                if distance_to_start <= 0.01:
                    print(f"In contact with downward staircase")
            else:
                # Check N = 6
                diff_s3_6 = (s3_constant - s3_vertical_distance) / 6
                if -0.17 <= diff_s3_6 <= -0.14:
                    downward_staircase_detected3 = True
                    d_3_s = 0.21017 + 0.3*5
                    d_i_h = calculate_horizontal_distance(s3_value, h_angle_s3_rad)
                    distance_to_start = calculate_distance_to_start_of_staircase(d_i_h,
d_S3_w, d_3_s)
                    last_value_down = downward_staircase_detected3
                    counter_down = 0
                    if distance_to_start >= 1:
                        print(f"Downward staircase in: {distance_to_start} meters")
                    if distance_to_start <= 0.01:
                        print(f"In contact with downward staircase")
                else:
                    # Check N = 7
                    diff_s3_7 = (s3_constant - s3_vertical_distance) / 7
                    if -0.17 <= diff_s3_7 <= -0.14:
                        downward_staircase_detected3 = True
                        d_3_s = 0.21017 + 0.3*6
                        d_i_h = calculate_horizontal_distance(s3_value, h_angle_s3_rad)
                        distance_to_start = calculate_distance_to_start_of_staircase(d_i_h,
d_S3_w, d_3_s)
                        last_value_down = downward_staircase_detected3
                        counter_down = 0
                        if distance_to_start >= 1:
                            print(f"Downward staircase in: {distance_to_start} meters")
                        if distance_to_start <= 0.01:
                            print(f"In contact with downward staircase")
                    else:
                        # Check N = 8
                        diff_s3_8 = (s3_constant - s3_vertical_distance) / 8
                        if -0.17 <= diff_s3_8 <= -0.14:
                            downward_staircase_detected3 = True
                            d_3_s = 0.21017 + 0.3*7
                            d_i_h = calculate_horizontal_distance(s3_value, h_angle_s3_rad)
                            distance_to_start = calculate_distance_to_start_of_staircase(d_i_h,
d_S3_w, d_3_s)
                            last_value_down = downward_staircase_detected3
```

```
                        counter_down = 0
                        if distance_to_start >= 1:
                            print(f"Downward staircase in: {distance_to_start} meters")
                        if distance_to_start <= 0.01:
                            print(f"In contact with downward staircase")
                    else:
                        # Check N = 9
                        diff_s3_9 = (s3_constant - s3_vertical_distance) / 9
                        if -0.17 <= diff_s3_9 <= -0.14:
                            downward_staircase_detected3 = True
                            d_3_s = 0.21017 + 0.3*8
                            d_i_h = calculate_horizontal_distance(s3_value,
h_angle_s3_rad)
                            distance_to_start =
calculate_distance_to_start_of_staircase(d_i_h, d_S3_w, d_3_s)
                            last_value_down = downward_staircase_detected3
                            counter_down = 0
                            if distance_to_start >= 1:
                                print(f"Downward staircase in: {distance_to_start} meters")
                            if distance_to_start <= 0.01:
                                print(f"In contact with downward staircase")
                        else:
                            # Check N = 10
                            diff_s3_10 = (s3_constant - s3_vertical_distance) / 10
                            if -0.17 <= diff_s3_10 <= -0.14:
                                downward_staircase_detected3 = True
                                d_3_s = 0.21017 + 0.3*9
                                d_i_h = calculate_horizontal_distance(s3_value,
h_angle_s3_rad)
                                distance_to_start =
calculate_distance_to_start_of_staircase(d_i_h, d_S3_w, d_3_s)
                                last_value_down = downward_staircase_detected3
                                counter_down = 0
                                if distance_to_start >= 1:
                                    print(f"Downward staircase in: {distance_to_start} meters")
                                if distance_to_start <= 0.01:
                                    print(f"In contact with downward staircase")
                            else:
                                downward_staircase_detected3 = last_value_down
                                counter_down += 1
                                if counter_down > downward_staircase_allowed_false:
                                    downward_staircase_detected3 = False
                                    downward_staircase_detected3 = last_value_down
                                    counter_down += 1
                                    if counter_down > downward_staircase_allowed_false:
                                        downward_staircase_detected3 = False

    return "upward_stairs" if upward_staircase_detected else "downward_stairs" if
downward_staircase_detected2 or downward_staircase_detected3 else None
```

```
# Define the heuristic function (Euclidean distance)
def heuristic(a, b):
    return sqrt((a[0] - b[0])**2 + (a[1] - b[1])**2)

# Create the graph
G = nx.Graph()

# Add nodes (waypoints) with positions
nodes = {
    # main road
    'A': (-44.8, 12.6),
    'B': (-34.8, 12.6),
    'C': (-24.8, 12.6),
    'D': (-14.8, 12.6),
    'E': (-4.8, 12.6),
    'F': (5.2, 12.6),
    'G': (9.8, 12.6),

    # main curve
    'H': (19, 12),
    'I': (22.6, 10.7),
    'J': (26.1, 8.29),
    'K': (30.4, 3.28),
    'L': (32.5, -1.96),
    'M': (33.3, -6.1),

    # after curve with crosswalk
    'N': (33.3, -18.6), #start sidewalk
    'O': (33.3, -30.6), #finish sidewalk
    'P': (33.3, -36.1),
    'Q': (33.3, -43.9),

    # shortcut
    'R': (-12.5, 16.7),
    'S': (-10.8, 10.9),
    'T': (-10, 2.15),
    'U': (-10.5, -3.85),
    'V': (-10.5, -13.85),
    'W': (-10.5, -16.1),

    # backside towards church
    'X': (-1, -18.6),
    'Z': (1, -18.6),
    'AA': (12.52, -18.6),
    'AB': (22.52, -18.6),
    'AC': (31.4, -18.6),
}

for node, pos in nodes.items():
    G.add_node(node, pos=pos, pedestrian=True)
```

```
# Function to add edge with calculated distance
def add_edge_with_distance(G, node1, node2):
    pos1 = G.nodes[node1]['pos']
    pos2 = G.nodes[node2]['pos']
    distance = heuristic(pos1, pos2)
    G.add_edge(node1, node2, weight=distance)

# Add edges with calculated weights (distances)
edges = [
    ('A', 'B'),
    ('B', 'C'),
    ('C', 'D'),
    ('D', 'E'),
    ('E', 'F'),
    ('F', 'G'),
    ('G', 'H'),
    ('H', 'I'),
    ('I', 'J'),
    ('J', 'K'),
    ('K', 'L'),
    ('L', 'M'),
    ('M', 'N'),
    ('N', 'O'),
    ('O', 'P'),
    ('P', 'Q'),
    ('E', 'R'),
    ('D', 'R'),
    ('R', 'S'),
    ('S', 'T'),
    ('T', 'U'),
    ('U', 'V'),
    ('V', 'W'),
    ('W', 'X'),
    ('X', 'Z'),
    ('Z', 'AA'),
    ('AA', 'AB'),
    ('AB', 'AC'),
    ('AC', 'N'),
]

for edge in edges:
    add_edge_with_distance(G, edge[0], edge[1])

# A* algorithm with pedestrian zone checking
def a_star_pedestrian(G, start, goal):
    open_set = []
    heapq.heappush(open_set, (0, start))
    came_from = {}
    g_score = {node: float('inf') for node in G.nodes}
```

```
    g_score[start] = 0
    f_score = {node: float('inf') for node in G.nodes}
    f_score[start] = heuristic(G.nodes[start]['pos'], G.nodes[goal]['pos'])

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

        if current == goal:
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            path.append(start)
            return path[::-1]

        for neighbor in G.neighbors(current):
            if not G.nodes[neighbor]['pedestrian']:
                continue  # Skip non-pedestrian nodes

            edge = G.edges[current, neighbor]
            tentative_g_score = g_score[current] + edge['weight']
            if tentative_g_score < g_score[neighbor]:
                came_from[neighbor] = current
                g_score[neighbor] = tentative_g_score
                f_score[neighbor] = g_score[neighbor] + heuristic(G.nodes[neighbor]['pos'],
G.nodes[goal]['pos'])
                heapq.heappush(open_set, (f_score[neighbor], neighbor))

    return None

# Calculate distance to the waypoint
def distance_to_waypoint(gps, x_des, y_des):
    position = gps.getValues()
    x_pos = position[0]
    y_pos = position[1]

    distance = sqrt((x_des - x_pos)**2 + (y_des - y_pos)**2)
    return distance

def normalize_angle(angle):
    while angle > pi:
        angle -= 2 * pi
    while angle < -pi:
        angle += 2 * pi
    return angle

def angle_to_waypoint(imu, x_des, y_des, x_pos, y_pos):
    orientation = imu.getRollPitchYaw()
    yaw_ETA = orientation[2]
```

```
    # Calculate the desired yaw angle to the waypoint
    yaw_des = atan2(y_des - y_pos, x_des - x_pos)

    # Normalize angles
    yaw_des = normalize_angle(yaw_des)
    yaw_ETA = normalize_angle(yaw_ETA)

    # Turn needed to orient the ETA towards the waypoint
    turn_angle = yaw_des - yaw_ETA
    turn_angle = normalize_angle(turn_angle)

    return turn_angle

def assign_next_waypoint(path, path_index):
    waypoint = path[path_index]
    x_des, y_des = G.nodes[waypoint]['pos']
    return x_des, y_des, (path_index == len(path) - 1)


def cluster_lidar_points(lidar_data, width_threshold=0.5, stability_threshold=0.05):
    clusters = []
    current_cluster = []

    for i, point in enumerate(lidar_data):
        if point[0] == float('inf') or point[1] == float('inf'):
            continue

        if not current_cluster:
            current_cluster.append(point)
        else:
            last_point = current_cluster[-1]
            if sqrt((point[0] - last_point[0])**2 + (point[1] - last_point[1])**2) < width_threshold:
                current_cluster.append(point)
            else:
                clusters.append(current_cluster)
                current_cluster = [point]

    if current_cluster:
        clusters.append(current_cluster)

    # Stability check for clusters
    stable_clusters = []
    for cluster in clusters:
        if len(cluster) > 1:
            stable = True
            for i in range(1, len(cluster)):
                if sqrt((cluster[i][0] - cluster[i - 1][0])**2 + (cluster[i][1] - cluster[i - 1][1])**2) >
stability_threshold:
                    stable = False
                    break
```

```
            if stable:
                # Identify leftmost and rightmost points after cluster is stable
                leftmost_point = max(cluster, key=lambda point: point[1])
              #  print("leftmost_point: ", leftmost_point)
                rightmost_point = min(cluster, key=lambda point: point[1])
                stable_clusters.append((cluster, leftmost_point, rightmost_point))

    return stable_clusters

def is_obstacle_in_path(lidar_data, x_pos, y_pos, x_des, y_des):
    clusters = cluster_lidar_points(lidar_data)

    path_vector = np.array([x_des - x_pos, y_des - y_pos])
    path_distance = np.linalg.norm(path_vector)
    path_unit_vector = path_vector / path_distance

    # Perpendicular vector for creating the corridor
    perpendicular_vector = np.array([-path_unit_vector[1], path_unit_vector[0]])

    # Create two parallel vectors 0.5 meters to each side of the path vector
    corridor_width = 0.5
    left_vector_start = np.array([x_pos, y_pos]) + corridor_width * perpendicular_vector
    left_vector_end = np.array([x_des, y_des]) + corridor_width * perpendicular_vector
    right_vector_start = np.array([x_pos, y_pos]) - corridor_width * perpendicular_vector
    right_vector_end = np.array([x_des, y_des]) - corridor_width * perpendicular_vector

    for cluster, leftmost_point, rightmost_point in clusters:
        for point in cluster:
            point_x, point_y = point[0] + x_pos, point[1] + y_pos  # Adjust point by robot's GPS
coordinates
            point_vector = np.array([point_x - x_pos, point_y - y_pos])
            point_distance = np.linalg.norm(point_vector)

            # Check if point is within the expanded corridor
            if point_distance <= path_distance:
                # Project the point onto the path vector to get the distance along the path
                projection_length = np.dot(point_vector, path_unit_vector)
                projection_point = np.array([x_pos, y_pos]) + projection_length *
path_unit_vector

                # Compute distance from the projection point to the actual point
                distance_to_path = np.linalg.norm(projection_point - np.array([point_x,
point_y]))

                # Check if the point is within the 1.0 meters width corridor
                if distance_to_path <= corridor_width:
                    print("Obstacle in the path")
                    return True, (point_x, point_y), (cluster, leftmost_point, rightmost_point)
    return False, None, None
```

```
def determine_follow_direction(leftmost_point, rightmost_point, center_point):
    if leftmost_point and rightmost_point and center_point:
        left_center_diff = heuristic(leftmost_point, center_point)
        right_center_diff = heuristic(rightmost_point, center_point)

        if left_center_diff > right_center_diff:
            print(f"Obstacle more to the right, turning left")
            return 'right'
        else:
            print(f"Obstacle more to the left, turning rigth")
            return 'right'
    return None

def adjust_path_around_obstacle(x_pos, y_pos, obs_x, obs_y, direction):
    if direction is None:
        raise ValueError(f"Invalid direction: {direction}")

    # Generate a temporary waypoint around the obstacle
    temp_waypoint_next_x = abs(obs_x - x_pos) - 3
    temp_waypoint_next_x *= math.copysign(1, obs_x - x_pos)
    temp_waypoint_next_x += x_pos

    if direction == 'right':
        temp_waypoint_next_y = abs(obs_y - y_pos) - 0.9
        temp_waypoint_next_y *= math.copysign(1, obs_y - y_pos)
        temp_waypoint_next_y += y_pos
    elif direction == 'left':
        temp_waypoint_next_y = abs(-obs_y + y_pos) + 0.5
        temp_waypoint_next_y *= math.copysign(1, obs_y - y_pos)
        temp_waypoint_next_y += y_pos

    temp_waypoint_after_x = abs(obs_x - x_pos) - 1
    temp_waypoint_after_x *= math.copysign(1, obs_x - x_pos)
    temp_waypoint_after_x += x_pos

    temp_waypoint_after_y = abs(obs_y - y_pos)
    temp_waypoint_after_y *= math.copysign(1, obs_y - y_pos)
    temp_waypoint_after_y += y_pos

    return temp_waypoint_next_x, temp_waypoint_next_y, temp_waypoint_after_x,
temp_waypoint_after_y

# Function to check distance sensor readings and determine obstacle direction
def check_distance_sensors(sensors):
    distances = [sensor.getValue() for sensor in sensors]
    s4_value = distances[3]
    h_angle_s4 = 13
    h_angle_s4_rad = math.radians(h_angle_s4)
    s4_vertical_distance = s4_value * math.sin(h_angle_s4_rad) + 0.998
```

```
    print(f"vertical Distance s4: {s4_vertical_distance}")

    if distances[0] < 1:
        s1_value = distances[0]
        h_angle_s1 = 24.75
        d_S1_w = 0.48416
        h_angle_s1_rad = math.radians(h_angle_s1)
        d_i_h = calculate_horizontal_distance(s1_value, h_angle_s1_rad)
        horizontal = d_i_h - d_S1_w
        print(f"Sensor 1 detected an obstacle at a distance of: ", {horizontal})
        return 'right', distances[0]

    elif distances[1] < 1.5:
        s2_value = distances[1]
        h_angle_s2 = 34
        d_S2_w = 0.53114
        h_angle_s2_rad = math.radians(h_angle_s2)
        d_i_h = calculate_horizontal_distance(s2_value, h_angle_s2_rad)
        horizontal = d_i_h - d_S2_w
        print(f"Sensor 2 detected an obstacle at a distance of", {horizontal})
        return 'right', horizontal

    elif distances[2] < 1.1:
        s3_value = distances[2]
        h_angle_s3 = 48
        d_S3_w = 0.57813
        h_angle_s3_rad = math.radians(h_angle_s3)
        d_i_h = calculate_horizontal_distance(s3_value, h_angle_s3_rad)
        horizontal = d_i_h - d_S3_w
        print(f"Sensor 2 detected an obstacle at a distance of", {horizontal})
        return 'left', distances[2]

    elif s4_vertical_distance < 1.7:
        s4_value = distances[3]
        h_angle_s4 = 15
        d_S4_w = 00.48416
        h_angle_s4_rad = math.radians(h_angle_s4)
        d_i_h = calculate_horizontal_distance(s4_value, h_angle_s4_rad)
        horizontal = d_i_h - d_S4_w
        print(f"Sensor 4 detected an overhead obstacle at a distance of", {horizontal})
        print(f"The obstacle is at a height of: {s4_vertical_distance}")
        return 'left', distances[3]

    return None, None

alert_active = False  # Add this flag to track if the alert is active

def movement_command(distance, is_last_waypoint, traffic_light_state):

    # Override stop command if alert is active
```

```
    if alert_active:
        return forward  # Continue moving forward if alert is active

    # Check for stairs using distance sensor data
    stairs_state = detect_stairs(s1_distance, s2_distance, s3_distance)
    if stairs_state == "upward_stairs":
        return forward  # Stop for upward stairs detection
    elif stairs_state == "downward_stairs":
        return forward  # Stop for downward stairs detection

    # Check traffic light state
    if traffic_light_state == 1:  # Red light
        print("Traffic light is red, stopping")
        return stop  # Stop
    elif traffic_light_state == 3:  # Green light
        print("Traffic light is green, safe to continue")
        return forward  # Move forward

    if is_last_waypoint:
        if distance <= 0.5:  # Close enough to stop
            print("Destination reached")
            return stop  # Stop
        else:
            return forward  # Move forward
    else:
        return forward  # Move forward

# Main loop
path = None
path_index = 0
obstacles = {}
temporary_waypoint_next = None
temporary_waypoint_after = None
follow_direction = None

max_turn_angle = pi / 2

# Ensure the camera is initially disabled
camera.disable()

# Flags to print messages only once
n_message_printed = False
o_message_printed = False
sidewalk_confirmed = False
red_light_detected = False
red_message_printed = False
green_light_detected = False
alert_message_printed = False

while robot.step(timestep) != -1:
```

```
        # Deactivate all sensors initially
        s1.disable()
        s2.disable()
        s3.disable()

        # Read the distance sensor data
        s1.enable(timestep)
        robot.step(timestep)
        s1_distance = s1.getValue()
        robot.step(int(T_i * 1000))  # Wait for the calculated time in milliseconds
        s1.disable()

        s2.enable(timestep)
        robot.step(timestep)
        s2_distance = s2.getValue()
        robot.step(int(T_i * 1000))  # Wait for the calculated time in milliseconds
        s2.disable()

        s3.enable(timestep)
        robot.step(timestep)
        s3_distance = s3.getValue()
        robot.step(int(T_i * 1000))  # Wait for the calculated time in milliseconds
        s3.disable()

    try:
        if path is None:
            # Get the current GPS coordinates
            position = gps.getValues()
            x_pos, y_pos = position[0], position[1]

            # Add the goal as a temporary node in the graph
            G.add_node('Goal', pos=(goal_x, goal_y), pedestrian=True)

            # Connect the goal node to the nearest waypoint
            nearest_node = min(nodes, key=lambda node: heuristic(G.nodes[node]['pos'],
(goal_x, goal_y)))
            add_edge_with_distance(G, 'Goal', nearest_node)

            # Find the nearest node to the current position as the start node
            start_node = min(G.nodes, key=lambda node: heuristic(G.nodes[node]['pos'],
(x_pos, y_pos)))

            # Find the path using A* algorithm
            path = a_star_pedestrian(G, start_node, 'Goal')
            if path:
                path_index = 0
                print(f"Path found: {path}")  # Debugging statement
            else:
                print("No path found")  # Debugging statement
```

```
            break

        if path_index < len(path):
            if temporary_waypoint_next and temporary_waypoint_after:
                # Use the temporary waypoints for obstacle avoidance
                distance_to_next = heuristic((x_pos, y_pos), temporary_waypoint_next)
                print(f"Distance to next temporary waypoint: {distance_to_next}")  # Debugging
statement
                if distance_to_next > 0.5:
                    # Go to the next temporary waypoint first
                    x_des, y_des = temporary_waypoint_next
                    is_last_waypoint = False  # Since this is a temporary waypoint, it's not the
final destination
                    print(f"Current position: {position}")
                    print(f"Temporary waypoint next to the obstacle created at:
{temporary_waypoint_next}")  # Debugging statement
                    print(f"Temporary waypoint after the obstacle created at:
{temporary_waypoint_after}")  # Debugging statement

                else:
                    # Go to the after temporary waypoint
                    x_des, y_des = temporary_waypoint_after
                    is_last_waypoint = False  # Since this is a temporary waypoint, it's not the
final destination
                    print(f"Temporary waypoint after the object created at:
{temporary_waypoint_after}")  # Debugging statement
                    temporary_waypoint_next = None  # Clear the next waypoint
                    temporary_waypoint_after = None  # Clear the after waypoint after reaching it

            else:
                # Get the current waypoint
                x_des, y_des, is_last_waypoint = assign_next_waypoint(path, path_index)
                print(f"Next waypoint: {x_des}, {y_des}")  # Debugging statement

                # Skip waypoints within 10 meters after detecting an obstacle
                if follow_direction is not None and heuristic((x_pos, y_pos), (x_des, y_des))
<=10:
                    path_index += 1
                    continue

            # Get the previous waypoint
            if path_index > 0:
                prev_x, prev_y = G.nodes[path[path_index - 1]]['pos']
            else:
                prev_x, prev_y = x_pos, y_pos

            # Get the current position and orientation
            position = gps.getValues()
            x_pos = position[0]
            y_pos = position[1]
```

```
        # Check if the robot has reached waypoint N
        if not sidewalk_confirmed and (round(x_pos, 1), round(y_pos, 1)) == (33.3, -17):
            camera.enable(timestep)

        # Check if the robot has reached waypoint O
        if not o_message_printed and (round(x_pos, 1), round(y_pos, 1)) == (33.3, -30.6):
            print("Crosswalk surpassed. Turning off camera.")
            camera.disable()
            o_message_printed = True

        # Get LIDAR data
        lidar_data = [(p.x, p.y) for p in lidar.getPointCloud() if p.x != float('inf') and p.y !=
float('inf')]

        # Calculate distance and angle to the waypoint
        distance = distance_to_waypoint(gps, x_des, y_des)
        angle = angle_to_waypoint(imu, x_des, y_des, x_pos, y_pos)

        # Constrain the turn angle to avoid excessive turning
        if abs(angle) > max_turn_angle:
            angle = max_turn_angle if angle > 0 else -max_turn_angle

        # Check for obstacles in the path
        obstacle_detected, closest_obstacle, cluster_data =
is_obstacle_in_path(lidar_data, x_pos, y_pos, x_des, y_des)

         # Check distance sensors for obstacle avoidance
          sensor_obstacle_direction, sensor_distance = check_distance_sensors([s1, s2,
s3, s4])

        if sensor_obstacle_direction:
            temporary_waypoint_next = None
            temporary_waypoint_after = None
            follow_direction = sensor_obstacle_direction
            obs_x, obs_y = x_pos + sensor_distance * cos(angle), y_pos + sensor_distance
* sin(angle)

        if follow_direction and not temporary_waypoint_next and not
temporary_waypoint_after:
            temp_next_x, temp_next_y, temp_after_x, temp_after_y =
adjust_path_around_obstacle(x_pos, y_pos, obs_x, obs_y, follow_direction)
            temporary_waypoint_next = (temp_next_x, temp_next_y)
            temporary_waypoint_after = (temp_after_x, temp_after_y)
            print(f"Adjusted temporary waypoints due to sensor: Next:
{temporary_waypoint_next}, After: {temporary_waypoint_after}")

        if obstacle_detected:
            # Calculate the distance to the closest obstacle
            distance_to_obstacle_x = closest_obstacle[0] - x_pos
```

```
            distance_to_obstacle_y = closest_obstacle[1] - y_pos

            # Calculate the Euclidean distance
            distance_to_obstacle = math.sqrt(distance_to_obstacle_x**2 +
distance_to_obstacle_y**2)
            print(f"Obstacle detected {distance_to_obstacle} meters away")  # Debugging
statement
            obstacle_distance = heuristic((x_pos, y_pos), closest_obstacle)
            if obstacle_distance <= 6:  # Check if obstacle is within 6 meters
                if follow_direction is None:
                    cluster, leftmost_point, rightmost_point = cluster_data
                    center_index = len(cluster) // 2
                    center_point = cluster[center_index]

                    # Determine the follow direction based on the obstacle edge points
                    follow_direction = determine_follow_direction(leftmost_point,
rightmost_point, center_point)
                    print(f"Follow direction: {follow_direction}")  # Debugging statement

                if follow_direction is not None and temporary_waypoint_next is None and
temporary_waypoint_after is None:
                    obs_x, obs_y = closest_obstacle

                    # Adjust path around the obstacle
                    temp_next_x, temp_next_y, temp_after_x, temp_after_y =
adjust_path_around_obstacle(x_pos, y_pos, obs_x, obs_y, follow_direction)
                    temporary_waypoint_next = (temp_next_x, temp_next_y)
                    temporary_waypoint_after = (temp_after_x, temp_after_y)
                    print(f"Adjusted temporary waypoints: Next: {temporary_waypoint_next},
After: {temporary_waypoint_after}")  # Debugging statement

                    # Skip the next A* algorithm waypoint
                    path_index += 1

        if camera.getSamplingPeriod() > 0:  # Check if the camera is enabled
            # Read the camera image data
            camera_image = camera.getImage()

            # Detect the traffic light state
            traffic_light_state = detect_traffic_light(camera_image)

            if not sidewalk_confirmed and (traffic_light_state == 1 or traffic_light_state ==
3):
                print("Starting sidewalk. Turning on camera.")
                print("The sidewalk is 10 meters long.")
                sidewalk_confirmed = True

            if traffic_light_state == 1:
                if green_light_detected and not alert_message_printed:
                    print("Alert: Traffic light changed from green to red!")
```

```
                alert_active = True
                alert_message_printed = True
                red_light_detected = False
                green_light_detected = False
            elif not red_message_printed and not alert_active:
                print("Traffic light is red, stopping")
                red_message_printed = True
            red_light_detected = True
        elif traffic_light_state == 3:
            if red_light_detected:
                print("Traffic light is green, safe to continue")
            green_light_detected = True
            red_light_detected = False
            red_message_printed = False
            alert_message_printed = False
            alert_active = False

    else:
        traffic_light_state = 0  # Default to no traffic light detected if camera is not
enabled


    # Proceed towards the waypoint
    turn_angle = angle

    # Send the turn angle, movement command, and velocity
    data_angle = struct.pack('f', turn_angle)
    emitter.send(data_angle)

    if alert_message_printed:
        movement = forward  # Continue moving forward on alert
    else:
        movement = movement_command(distance, is_last_waypoint,
traffic_light_state)

    movement = movement_command(distance, is_last_waypoint, traffic_light_state)
    data_movement = struct.pack('B', movement)
    emitter.send(data_movement)

    velocity_str = f"{velocity:05.3f}"  # Format to 5 characters
    data_velocity = velocity_str.encode('utf-8')
    emitter.send(data_velocity)

    if movement == stop and is_last_waypoint:  # Destination reached and is the final
waypoint
        print("Destination reached.")  # Debugging statement
        break  # Stop the main loop

    if distance <= 0.2:
        if temporary_waypoint_next:
```

```
                # Check if the robot has reached the next temporary waypoint
                if heuristic((x_pos, y_pos), temporary_waypoint_next) <= 0.5:
                    temporary_waypoint_next = None  # Clear the next temporary waypoint
once reached
            elif temporary_waypoint_after:
                # Check if the robot has reached the after temporary waypoint
                if heuristic((x_pos, y_pos), temporary_waypoint_after) <= 0.5:
                    temporary_waypoint_after = None  # Clear the after temporary waypoint
once reached
                    follow_direction = None  # Reset follow direction
            else:
                # Move to the next waypoint if close enough and no temporary waypoints
remain
                path_index += 1
        else:
            print("All waypoints visited.")  # Debugging statement
            break  # All waypoints have been visited

    except Exception as e:
        print(f"Error occurred: {e}")
        continue  # Ensure the loop continues even if an error occurs
```