Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Asistencia remota de personas de movilidad reducida mediante la monitorización y control de una silla de ruedas inteligente mediante una interfaz web

Remote assistance for people with reduced mobility by monitoring and controlling an intelligent wheelchair using a web interface

Cristian Jonay Ramos González

La Laguna, 5 de julio de 2019

D. **Néstor Morales Hernández**, con N.I.F. 54048001-W profesor adscrito al Departamento de Ingeniería Informática de la Universidad de La Laguna, como tutor

D. **Leopoldo Acosta Sánchez**, con N.I.F. 42165911-B profesor titular adscrito al Departamento de Ingeniería Informática de la Universidad de La Laguna, como cotutor

C E R T I F I C A (N)

Que la presente memoria titulada:

”*Asistencia remota de personas de movilidad reducida mediante la monitorización y control de una silla de ruedas inteligente mediante una interfaz web*”

ha sido realizada bajo su dirección por D. **Cristian Jonay Ramos González**, con N.I.F. 45854491-N.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 5 de julio de 2019

Agradecimientos

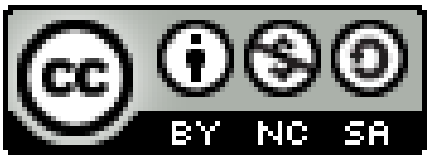
A mi familia, sobre todo a mi madre que siempre está ahí.

A mis amigos, que saben aguantar mis rarezas.

Y sobre todo a mi tutor Néstor por haber sacado tiempo para mí prácticamente todas las semanas y haberme ayudado tanto, sin él no hubiera podido completar este proyecto.

!Muchas gracias!

Licencia



Resumen

El objetivo de este trabajo ha sido la implementación del control y monitorización de una silla de ruedas autónoma, a través de una interfaz web para el cuidador y otra interfaz para un usuario de la silla. Se ha elegido una interfaz web para que la herramienta sea multiplataforma y sea accesible desde cualquier punto con acceso a internet. La herramienta a desarrollar está orientada a personas a cargo de pacientes con algún tipo de discapacidad, potenciales usuarios de la silla. Un objetivo adicional ha sido la implementación de un módulo que permita hacer una videollamada entre un usuario cuidador y un usuario de la silla.

Palabras clave: ROS, Robótica, Aplicaciones Web, WebRTC, interfaces

Abstract

The objective of this work has been the implementation of the control and monitoring of an autonomous wheelchair, through a web interface for the caregiver and another interface for a user of the chair. We chose using a web interface so that the tool is cross-platform and accessible from any point with internet access. The tool to be developed is aimed at people in charge of patients with some type of disability, potential users of the chair. An additional objective has been the implementation of a module that allows a video call between a carer user and a user of the chair.

Keywords: ROS, Robotics, Web applications, WebRTC, interfaces

Índice general

1. **Introducción** **1**
   - 1.1. Motivación . . . 1
   - 1.2. Objetivos . . . 2
   - 1.3. Estado del arte . . . 2
   - 1.4. Estructura del documento . . . 3
2. **Tecnologías empleadas** **4**
   - 2.1. Robot Operating System . . . 4
     - 2.1.1. Robot Web Tools . . . 5
     - 2.1.2. Web Video Server . . . 6
   - 2.2. WebRTC . . . 6
   - 2.3. Janus WebRTC Server . . . 7
     - 2.3.1. Plugin videoroom de Janus . . . 8
   - 2.4. NodeJS . . . 8
   - 2.5. NGINX . . . 9
   - 2.6. SQLite . . . 9
   - 2.7. Tecnologías web . . . 9
3. **Desarrollo del proyecto** **10**
   - 3.1. Integración con ROS . . . 11
     - 3.1.1. Visualización del mapa . . . 12
     - 3.1.2. Recibir otros tópicos . . . 13
     - 3.1.3. Publicación de tópicos . . . 16
     - 3.1.4. Visualización de tópicos imagen . . . 17
   - 3.2. Integración con Janus . . . 17
     - 3.2.1. Plugin Videoroom . . . 18
   - 3.3. Desarrollo de la interfaz de usuario . . . 21
     - 3.3.1. Interfaz del cuidador . . . 21
     - 3.3.2. Interfaz del usuario de la silla . . . 25
4. **Ejecución del prototipo** **27**
   - 4.1. Configuración del entorno . . . 27
     - 4.1.1. Rosbridge . . . 27
     - 4.1.2. Janus . . . 27
     - 4.1.3. Web Video Server . . . 28
     - 4.1.4. Nginx . . . 28
     - 4.1.5. Aplicación web . . . 29
   - 4.2. Lanzamiento . . . 29

4.3. Manual del usuario 30
4.3.1. Manual del cuidador 30
4.3.2. Manual del usuario de la silla 31
5. Conclusiones y líneas futuras 33
6. Summary and Conclusions 34
7. Presupuesto 35

Índice de Figuras

1.1. Silla del proyecto perenquén de la ULL 2

2.1. Comunicación en ROS 4
2.2. Estructura de un mensaje en ROS 5
2.3. Funcionamiento del plugin Videoroom 8
2.4. Formato de la petición para crear una sala en Videoroom 8

3.1. Arquitectura del proyecto 10
3.2. Listado de nodos cuando se está ejecutando el simulador 11
3.3. Visualización del mapa en la web 12
3.4. Comando rostopic list 13
3.5. Mapa con el marcador de la silla 15
3.6. Mapa con el marcador del camino que seguirá la silla 15
3.7. Movimiento de la silla al hacer doble click en el mapa 16
3.8. Simulador de la silla con la silla en movimiento 16
3.9. Respuesta de éxito al unirse a una sala 18
3.10. Elementos de la interfaz web del cuidador. 21
3.11. Gráficas de algunos tópicos. 24
3.12. Videoconferencia cuidador. 25
3.13. Elementos de la interfaz web del usuario de la silla. 25
3.14. Videoconferencia en la interfaz de la silla. 26

4.1. Interfaz web del cuidador. 30
4.2. Interfaz del usuario de la silla. 31

Índice de Tablas

7.1. Presupuesto. 35

Capítulo 1

Introducción

Este trabajo de fin de grado trata de la implementación de una interfaz web que permita controlar una silla de ruedas autónoma. Una silla de ruedas autónoma es una silla capaz de moverse hacia un lugar objetivo de una zona interior o exterior sin la intervención de un humano.

Además se ha implementado herramientas de monitorización de la silla y su usuario, y comunicación entre el usuario de la silla y un potencial cuidador.

1.1. Motivación

La herramienta a desarrollar está orientada a personas a cargo de pacientes con algún tipo de discapacidad, potenciales usuarios de la silla, y para los familiares de personas con movilidad reducida. Otro caso interesante es usar estas sillas en hospitales. Esto no sólo serviría para facilitar tareas al personal médico, pudiendo conectarse con el usuario y monitorizarlo sin tener que ir su habitación sino que serviría al nuevo usuario para moverse por el hospital sin conocerlo. Moverse por lugares desconocidos es especialmente complicado para usuarios con movilidad reducida, por lo que un sistema como el que se plantea sería de gran ayuda.

La herramienta es una plataforma web por lo que se puede usar desde cualquier punto con conexión a internet y desde cualquier plataforma.

Además, se puede añadir una herramienta de *eye tracking* para que el usuario de la silla la pueda manejar la vista, u otras interfaces hombre-máquina, sin modificar la aplicación.

1.2. Objetivos

Entre los objetivos de monitorización están los de recibir información en línea de la posición de la silla y sensores acoplados a la misma, incluyendo cámaras y sensores relacionados con la salud del paciente. Esta información se estará disponible para la persona al cargo a través su interfaz web.

También a través de esta interfaz se permitirá el control de la misma, reposicionando la silla dentro de la vivienda.

Por último, se ha implementado un módulo para hacer videoconferencia entre el cuidador y el usuario de la silla.

1.3. Estado del arte

Distintos avances de la tecnología y la inteligencia artificial han permitido que surjan las sillas de ruedas autónomas. En 2017 se implementó una de las primeras siendo (8) fruto de la alianza entre la Fundación Nacional de Investigación de Singapur (NRF) y el Instituto de Tecnología de Massachusetts (MIT) que llevaron la tecnología de los coches de conducción autónoma a las sillas de ruedas.

Existen diversos proyectos relativos al desarrollo de estas sillas de ruedas autónomas. En la Universidad de La Laguna (ULL), por ejemplo, existe el proyecto perenquén. El objetivo de este proyecto es (13) diseñar e implementar una silla de ruedas auto-guiada inteligente (figura 1.1).

Figura 1.1: Silla del proyecto perenquén de la ULL

Además, en los últimos tiempos, han surgido un gran número de proyectos destinados al desarrollo de nuevas formas de controlar una silla de ruedas. Por ejemplo, en 2010 un equipo de investigadores dirigido por el profesor José del R. Millán en el Instituto Federal de Tecnología de Suiza (EPFL), Lausana, Suiza, desarrolló una silla de ruedas que se puede controlar a través de una interfaz cerebro-computadora. El foco de su investigación fue el uso directo de las señales del cerebro humano para controlar dispositivos e interactuar con el medio ambiente. También existen se han desarrollado interfaces para controlar sillas con la mirada o incluso con las emociones.

Estos proyectos pretenden mejorar la vida de las personas con movilidad reducida, reduciendo su dependencia, así como facilitar el trabajo a sus familiares, cuidadores o personal médico.

1.4. Estructura del documento

Este documento sigue la siguiente estructura de capítulos:

- **2. Tecnología empleadas.** Se describe brevemente cada una de las tecnologías de las que se ha hecho uso en el desarrollo de este trabajo de fin de grado (TFG).
- **3. Desarrollo del proyecto.** Se describen en detalle los pasos seguidos para la implementación del prototipo de forma ordenada.
- **4. Ejecución del prototipo.** Se describe cómo configurar el entorno y lanzar el prototipo paso a paso.
- **5. Conclusiones y líneas futuras** Se plantean unas conclusiones así como unos pasos a seguir o características a añadir en el futuro a este proyecto.
- **6. Resumen y conclusiones.** Resumen y conclusiones realizadas (en inglés).
- **7. Presupuesto.** Se plantea un presupuesto aproximado de la herramienta y el trabajo realizado.

Capítulo 2

Tecnologías empleadas

Una de las partes que más complejidad ha añadido al proyecto es la necesidad de integrar múltiples tecnologías, tanto de interacción directa con la silla (de ahí la necesidad de trabajar con ROS), como de los distintos microservicios que forman el backend de la aplicación (servicio de traducción de los tópicos de ROS, servidor de signaling WebRTC, etc.). Estas tecnologías se listan a continuación.

2.1. Robot Operating System

La tecnología base de este proyecto ha sido Robot Operating System (ROS), ya que es la herramienta con la que está desarrollada el control de la silla de la que se ha hecho la interfaz. ROS (6) se trata de un framework que ofrece un conjunto de herramientas, librerías, y convenciones para facilitar la programación de comportamientos para robots complejos.

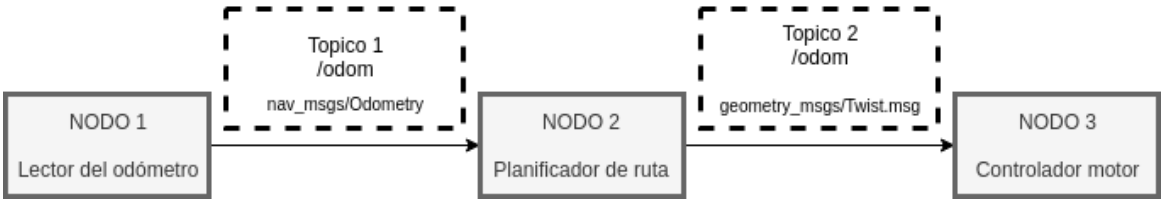
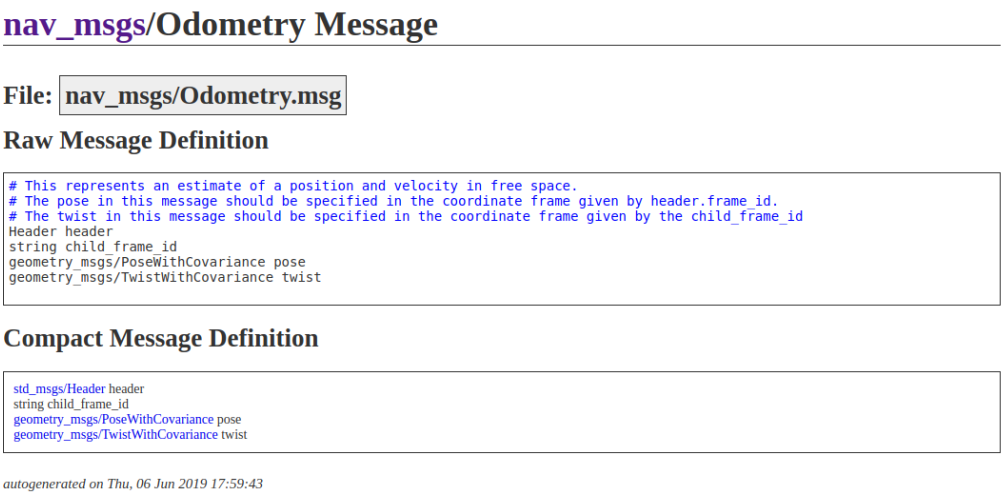


Figura 2.1: Comunicación en ROS

ROS está diseñado para ser modular. El software que controla un robot se divide en módulos software, llamados nodos, donde cada uno se encarga de realizar una tarea y luego estos nodos se comunican entre siguiendo una arquitectura Publicador-Suscriptor.

Tomemos el ejemplo de la figura 2.1 (15). El nodo NODO 1 se trata de un componente software que se encarga de leer el odómetro de las ruedas para saber la posición en la que se encuentra el robot. Esta posición la

publicará constantemente el nodo a través de lo que se llama un tópico. Los tópicos son nombres que se le dan a cada canal por el que un nodo envía mensajes. Cada mensaje tiene un formato (figura 2.2) que los demás nodos necesitarán conocer. Se necesitan estos tópicos puesto que cada nodo puede publicar varios mensajes con distinto propósito, es decir a múltiples tópicos, incluso si los mensajes son del mismo formato. Además, varios nodos pueden publicar al mismo tópico. El nodo NODO 2, que se encarga de planificar los movimientos necesarios para llegar a un determinado punto, se suscribirá al tópico /odom para saber la posición actual del robot y enviará al nodo NODO 3 los comandos de movimiento necesarios para llegar al objetivo. Por último el nodo NODO 3 mandará al robot a moverse de determinada forma.



nav_msgs/Odometry Message

File: nav_msgs/Odometry.msg

Raw Message Definition

```
# This represents an estimate of a position and velocity in free space.
# The pose in this message should be specified in the coordinate frame given by header.frame_id.
# The twist in this message should be specified in the coordinate frame given by the child_frame_id
Header header
string child_frame_id
geometry_msgs/PoseWithCovariance pose
geometry_msgs/TwistWithCovariance twist
```

Compact Message Definition

```
std_msgs/Header header
string child_frame_id
geometry_msgs/PoseWithCovariance pose
geometry_msgs/TwistWithCovariance twist
```

autogenerated on Thu, 06 Jun 2019 17:59:43

Figura 2.2: Estructura de un mensaje en ROS

Esta modularidad fomenta la colaboración y hace que existan una gran cantidad de paquetes con distintas funcionalidades hechos y publicados por la comunidad listos para integrar en cualquier proyecto.

2.1.1. Robot Web Tools

Robot Web Tools se trata de una colección de librerías y herramientas que nos sirven para interactuar con diferentes middlewares que se ejecutan en robots como el ya nombrado ROS. De dichas librerías, para este proyecto hemos usado concretamente las librerías ROSLIBJS y ROS2DJS.

ROSLIBJS

Se trata de una librería usada para interactuar con ROS desde el navegador web. Esta librería nos proporciona la capacidad de publicar y

suscribirse a los tópicos de nuestra silla. Para ello la librería hace uso de WebSockets para conectarse con el servidor rosbridge. El servidor rosbridge permite que las páginas web se comuniquen con ROS utilizando el protocolo rosbridge, que es un nodo de ROS. El servidor Rosbridge crea una conexión WebSocket y convierte los mensajes JSON(JavaScript Object Notation) recibidos desde el WebSocket a llamadas ROS y al inverso, convierte las respuestas de ROS a JSON para enviarlas por el WebSocket.

De este modo, los tópicos que están disponibles en ROS podrán ser consultados por cualquier aplicación Web (o cualquier tecnología capaz de abrir un websocket), y podremos publicar sobre los tópicos desde dicha aplicación.

ROS2DJS

Esta biblioteca (17) proporciona un administrador de visualización 2D para ROS. Usando esta biblioteca, podemos visualizar mapas 2D en un navegador web. Esta librería está construida por encima de ROSLIBJS y EaselJS, es decir, se necesita de estas para que funcione, y no sólo nos permitirá visualizar el mapa sino funciones para añadir marcadores al mapa o dibujar los mensajes del tipo `nav_msgs/Path` que nos permiten visualizar el camino que sigue la silla.

2.1.2. Web Video Server

Nodo ROS que implementa un servidor HTTP para la transmisión de tópicos imágenes. Web Video Server (16) abre un puerto local y espera las solicitudes HTTP entrantes. Tan pronto como se solicita un flujo de vídeo de un tópico imagen ROS a través de HTTP, se suscribe al tópico correspondiente y crea una instancia del codificador de vídeo. Los paquetes de vídeo en bruto codificados se sirven luego al cliente.

2.2. WebRTC

WebRTC se trata de un framework abierto que permite comunicaciones en tiempo real a través del navegador web. WebRTC incluye las piezas fundamentales para las comunicaciones como son red, audio y vídeo. WebRTC está habilitado en navegadores como Opera, Google Chrome y Firefox, por lo que no se necesitan ni plugins ni instalaciones para su uso. WebRTC (18) ofrece tres APIs u objetos principales MediaStream RTCPeerConnection y

RTCDataChannel. Para este proyecto no se hizo uso del canal de datos así que nos centraremos en los otros.

Usamos MediaStream para recoger el audio y/o vídeo del usuario. MediaStream (18) representa un único stream en el que se encuentran múltiples pistas, como pueden ser una de audio que proviene de un micrófono y otra de vídeo que proviene de la cámara web, y que están sincronizadas.

Por otra parte, RCTPeerConnection (18) no sólo establece una comunicación entre dos pares, sino que también permite el envío de los datos necesarios (14) de ambos pares:

- Control de sesión.
- Mensajes de error.
- Meta datos multimedia tales como codecs, ancho de banda y tipos de multimedia.
- Datos clave, usados para establecer una conexión segura..
- Datos de red, puerto y dirección IP pública.

El proceso de intercambio de esta información se llama signaling y no está estandarizado. Se necesita construir un servidor específico de cada aplicación para mandar y recibir los mensajes. Afortunadamente, existen distintas tecnologías Open Source que se pueden encargar de este proceso. Nosotros no hemos implementado un servidor propio, sino que hemos usado Janus, un servidor WebRTC de propósito general.

2.3. Janus WebRTC Server

Janus (10) es un servidor WebRTC desarrollado por Meetecho concebido para ser de propósito general. Así pues, no proporciona ninguna otra funcionalidad que no sea la implementación de los medios para configurar una comunicación multimedia WebRTC a través del navegador, intercambiar mensajes JSON con él y transmitir RTP / RTCP y mensajes entre navegadores y la lógica de la aplicación del lado del servidor. Sin embargo, Janus ofrece funcionalidades adicionales a través de plugins.

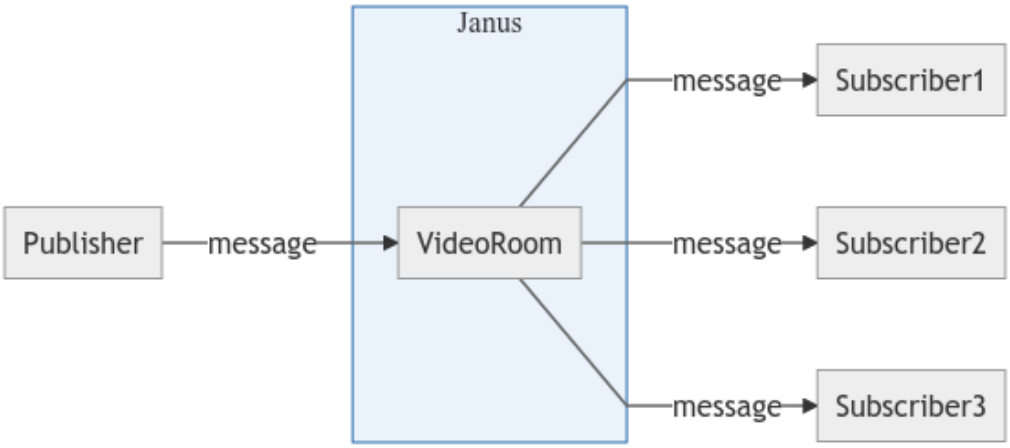


Figura 2.3: Funcionamiento del plugin Videoroom

2.3.1. Plugin videoroom de Janus

Este plugin (11) implementa una Selective Forwarding Unit(SFU) para la videoconferencia. Una SFU recibe múltiples transmisiones multimedia y decide a quién se las debe enviar. Videoroom usa el patrón publicador-suscriptor (figura 2.3) y tiene una API a la que se le pueden hacer múltiples peticiones en formato JSON(figura 2.4) como `create`, `join`, o `publish`, etc.

```
{
        "request" : "create",
        "room" : <unique numeric ID, optional, chosen by plugin if missing>,
        "permanent" : <true|false, whether the room should be saved in the config file, default=false>,
        "description" : "<pretty name of the room, optional>",
        "secret" : "<password required to edit/destroy the room, optional>",
        "pin" : "<password required to join the room, optional>",
        "is_private" : <true|false, whether the room should appear in a list request>,
        "allowed" : [ array of string tokens users can use to join this room, optional],
        ...
}
```

Figura 2.4: Formato de la petición para crear una sala en Videoroom

2.4. NodeJS

NodeJS se trata de un entorno orientado a eventos para la ejecución de JavaScript en lado del servidor. NodeJS implementa una base en su core y permite la instalación de paquetes a través del gestor de paquetes npm (node package manager). Para el desarrollo de nuestro proyecto hemos usado, entre otros, los paquetes Express y Socket.io.

Express (2) es una infraestructura de aplicaciones web Node.js que proporciona un conjunto de características para las aplicaciones web. Nos sirve para responder las peticiones HTTP del cliente con sus métodos.

Socket.IO (7), por otro lado, permite la comunicación en tiempo real, bidireccional y basada en eventos.

2.5. NGINX

NGINX (9) es un software de código abierto para la implantación servidores web, proxy inverso, almacenamiento en caché, balanceo de carga, transmisión de medios y más. En nuestro proyecto se usó como proxy inverso para poder servir nuestra web con HTTPS.

2.6. SQLite

Sistema gestor de base de datos que al contrario que otros sistemas no se implementa en un servidor aparte, sino que este gestor escribe directamente sobre uno o varios archivos. Usa sentencias SQL, y su sencillez y hicieron que fuera muy apropiado para nuestro proyecto.

2.7. Tecnologías web

Al tratarse, nuestro proyecto, de una aplicación web se han usado la tecnologías básicas de la web: HTML, Javascript y CSS. Además de estas tecnologías se han usado la librería jQuery, para la manipulación del DOM, Google Charts, librería para la visualización de gráficas y Materialize como framework CSS, que usa Material Design.

Capítulo 3

Desarrollo del proyecto

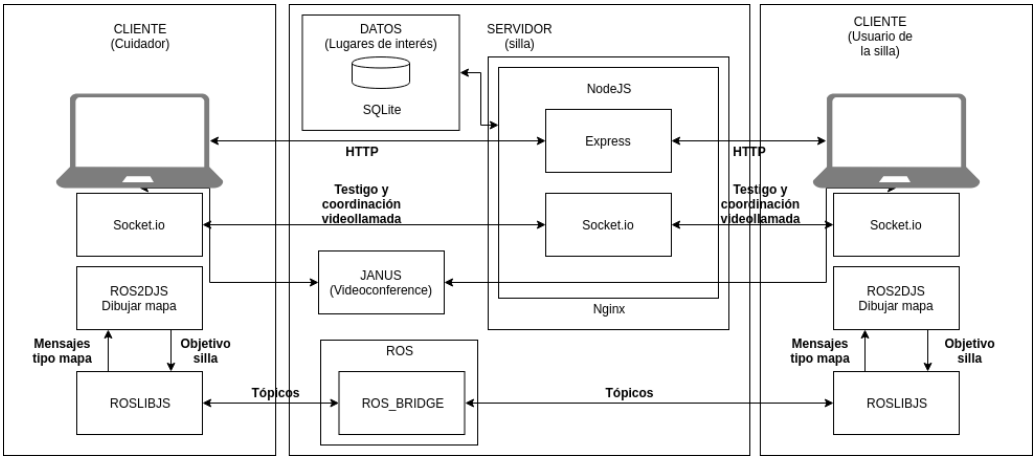


Figura 3.1: Arquitectura del proyecto

Este prototipo consta (figura 3.1) de dos interfaces una para el usuario de la silla y otra para el cuidador. El cuidador tiene capacidad de controlar la silla de forma remota, y de monitorizar ciertos aspectos. Para la mayoría de estas funcionalidades requiere solicitar previamente un token, que le da el control de la silla (evita que haya más de un usuario controlando la silla). Cuando tiene el control, el cuidador puede manejar la silla, establecer puntos de interés y hacer videollamadas con el usuario de la silla. También puede monitorizar el estado de los distintos sensores instalados en la silla, incluyendo las cámaras. Desde el lado de la silla, es posible navegar hacia los puntos de interés establecidos por el cuidador y hacer videollamadas.

Todo esto es controlado por el backend, basado en NodeJS, y que se encuentra detrás de un servidor Nginx, que hace de proxy inverso.

Otros servicios, como Janus, proporcionan el control de la parte de videoconferencias; o ros - con los nodos `ros_bridge` y `ros_video_server` - permite la interacción con los tópicos. Esto nos permite tener un backend distribuido con pequeños servidores muy especializados siguiendo

una arquitectura próxima a la arquitectura de microservicios. Para el almacenamiento de las configuraciones se usó SQLite.

3.1. Integración con ROS

En este apartado describiremos el desarrollo de la integración de nuestra aplicación con ROS. Lo primero que tuvimos que hacer fue instalar ROS es nuestro equipo, la versión que usamos fue Kinetic en Linux. Se ha de decir que para el desarrollo del proyecto se usó un simulador del sistema ROS que corre en la silla. Ya con ROS instalado, y el simulador corriendo se pudo empezar con el desarrollo de la parte que se integra con ROS, con los siguientes objetivos:

- Recibir y visualizar el mapa.
- Recibir otros tópicos: posición de la silla, camino planeado, velocidad, etc
- Publicar tópicos: mover y parar silla.
- Recibir y visualizar tópicos imagen.

Aunque el primer paso fue entender el funcionamiento de la silla: qué nodos y tópicos tiene y cuál es su función. Para ello, nos fueron útiles algunos comando de ROS, ejecutados cuando se está ejecutando el simulador, como son rosnode list (figura 3.2), que nos muestra los nodos que se están ejecutando, rosnode info [nombre del nodo], que nos muestra información sobre el nodo(Tópicos que publica, tópicos a los que esta suscrito, etc) o rostopic info [nombre del tópico], que nos muestra información del tópico (nodos que lo publican/suscriben y nombre del mensaje).

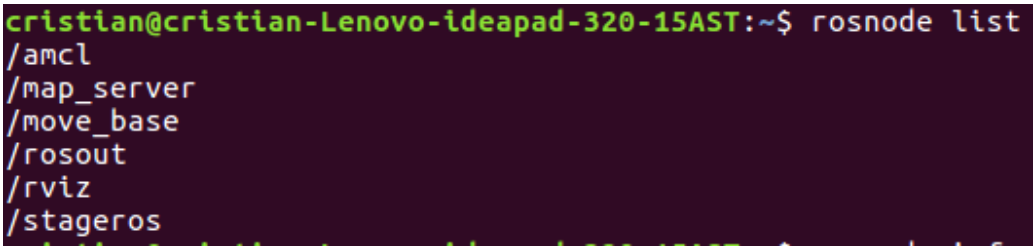


Figura 3.2: Listado de nodos cuando se está ejecutando el simulador

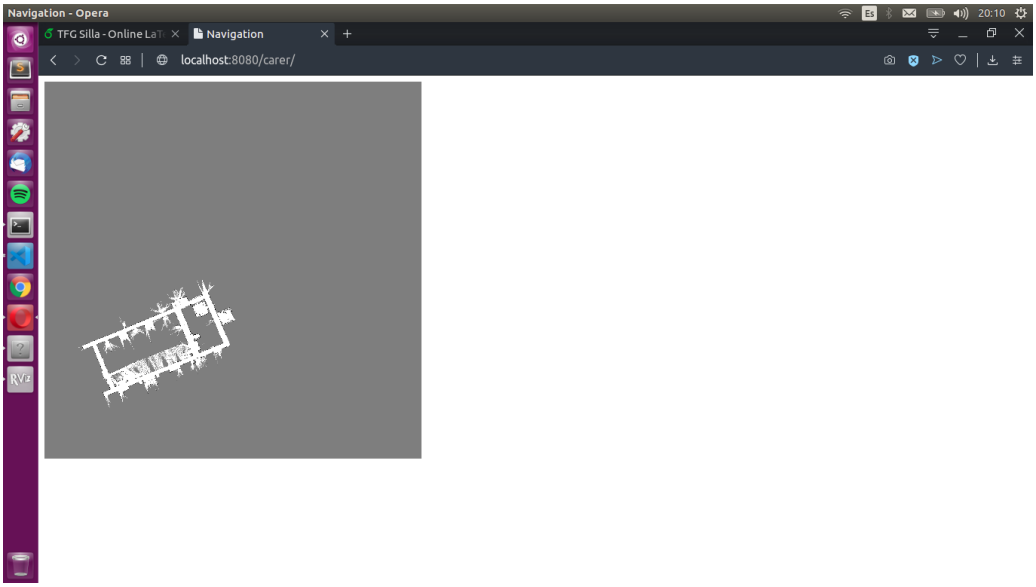


Figura 3.3: Visualización del mapa en la web

3.1.1. Visualización del mapa

Para visualización del mapa, necesitamos la biblioteca ROS2DJS, así como sus dependencias.

```
<script src="https://static.robotwebtools.org/EaselJS/current/easeljs.js"></script>
<script src="https://static.robotwebtools.org/EventEmitter2/current/eventemitter2.min.js"></script>
<script src="https://static.robotwebtools.org/roslibjs/current/roslib.js"></script>
<script src="js/ros2d.js"></script>
```

Por otra parte, necesitaremos instalar y ejecutar el nodo `ros_bridge` para comunicarnos con ROS que correrá en un puerto especifico.

```
> roslaunch rosbridge_server rosbridge_websocket.launch
```

Una vez que tengamos `ros_bridge` creamos un script donde nos conectaremos con él usando la clase `Ros` de ROSLIBJS.

```
class Navigation {
    constructor(url_ros) {
        this.ros = new ROSLIB.Ros({
            url : url_ros//Conexion a rosbridge
        });

        this._initialize_topics();
```

Para visualizar el mapa usamos un objeto de la clase `Viewer` de ROS2DJS que renderizará el mapa en un canvas HTML5 que añadirá en el elemento `<div>` con el ID que le especifiquemos. El objeto `OccupancyGridClient` escuchara el tópico `/map` de la conexión con rosbridge creado anteriormente y cuando haya algún cambio enviará el evento `change`. Podemos escuchar este evento con el método `on('change')` del `OccupancyGridClient` y hacer las operaciones necesarias.

```
class Navigation {
    constructor(url_ros) {
        this.ros = new ROSLIB.Ros({
            url : url_ros//Conexion a rosbridge
        });

        this._initialize_topics();
```

```
    }

    set_map(divID){
        this.viewer2D = new ROS2D.Viewer({
           divID : divID,
           width : 500,
           height : 500
        });

        this.gridClient = new ROS2D.OccupancyGridClient({
          ros : this.ros,
          rootObject : this.viewer2D.scene
        });

        this.gridClient.on('change', function() {
            viewer.scaleToDimensions(gridClient.currentGrid.width, gridClient.currentGrid.height)
        }.bind(this));
    }
```

Así pues, sólo creamos el `<div>` en el html, con una ID que le pasamos a la función `set_map(id)`. Podemos visualizar un ejemplo en la figura 3.3.

```
var map_navigation;

map_navigation = new Navigation('ws://127.0.0.1:9090');
map_navigation.set_map("twod-map");
```

Esta visualización del mapa la realizamos en ambas interfaces, tanto la del cuidador como la del usuario de la silla.

3.1.2. Recibir otros tópicos

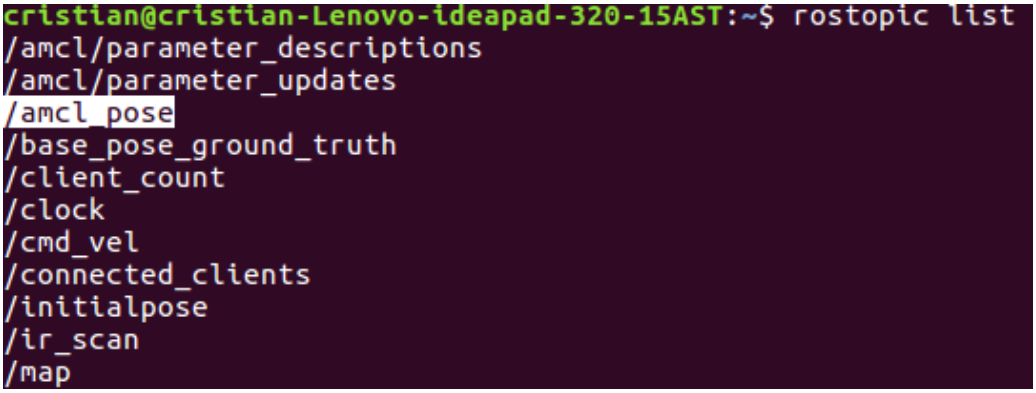


Figura 3.4: Comando rostopic list

Para suscribirnos a los tópicos primero tenemos que inicializarlos usando la clase `Topic` de ROSLIBJS, a su constructor le pasaremos como parámetros la conexión con `ros_bridge`, el nombre del tópico y el tipo de mensaje.

En este ejemplo nos suscribimos al tópico `/acml_pose` (figura 3.4) que nos devuelve la posición de la silla.

```
class Navigation {
    constructor(url_ros) {
        this.ros = new ROSLIB.Ros({
            url : url_ros//Conexion a rosbridge
        });

        this._initialize_topics();
    }

    set_map(divID){
    }
```

```
    _initialize_topics(){
        this.poseTopic = new ROSLIB.Topic({
            ros          : this.ros,
            name         : '/amcl_pose',//Topico
            messageType : 'geometry_msgs/PoseWithCovarianceStamped' //tipo del mensaje
        });
    }
```

Una vez inicializado, podemos usar los métodos `subscribe` y `publish`. Como queremos recibirlos, usamos el método `subscribe` al que le pasamos como parámetro una función callback, que se llamará cuando nos llegue algún mensaje del tópico.

```
map_navigation.velTopic.subscribe(function(message){
    console.log(message);
});
```

Este mismo procedimiento lo seguimos con los distintos tópicos de la silla a los que nos queramos suscribir.

Dibujando el tópico posición

Una vez suscritos al tópico `/amcl_pose`, que nos da la posición y orientación de la silla, ROS2DJS nos ofrece una funcionalidad para dibujar esta posición y orientación en el mapa que ya estamos visualizando. Para ello instanciamos un objeto de la clase NavigationArrow. Cada vez que nos llegue un mensaje del tópico que nos da la posición, dibujaremos esta NavigationArrow en el mapa (figura 3.5).

```
class Navigation {
    constructor(url_ros) {
    }

    set_map(divID){
    }

    _initialize_topics(){
    }

    _set_marker_on_map(){
        this.robotMarker = new ROS2D.NavigationArrow({
          size : 12,
          strokeSize : 1,
          fillColor : createjs.Graphics.getRGB(255, 128, 0, 1),
          pulse : true
        });
        this.initScaleSet = false;
        this.gridClient.rootObject.addChild(this.robotMarker);
        this.poseTopic.subscribe(function(message) {

          this.position = message.pose.pose;

          this.robotMarker.x = this.position.position.x;
          this.robotMarker.y = -this.position.position.y;

          if (!this.initScaleSet) {
            this.robotMarker.scaleX = 1.0 / this.viewer2D.scene.scaleX;
            this.robotMarker.scaleY = 1.0 / this.viewer2D.scene.scaleY;
            this.initScaleSet = true;
          }
          this.robotMarker.rotation = this.viewer2D.scene.rosQuaternionToGlobalTheta(message.pose.pose.orientation);
          this.robotMarker.visible = true;
        }.bind(this));
      }
}
```

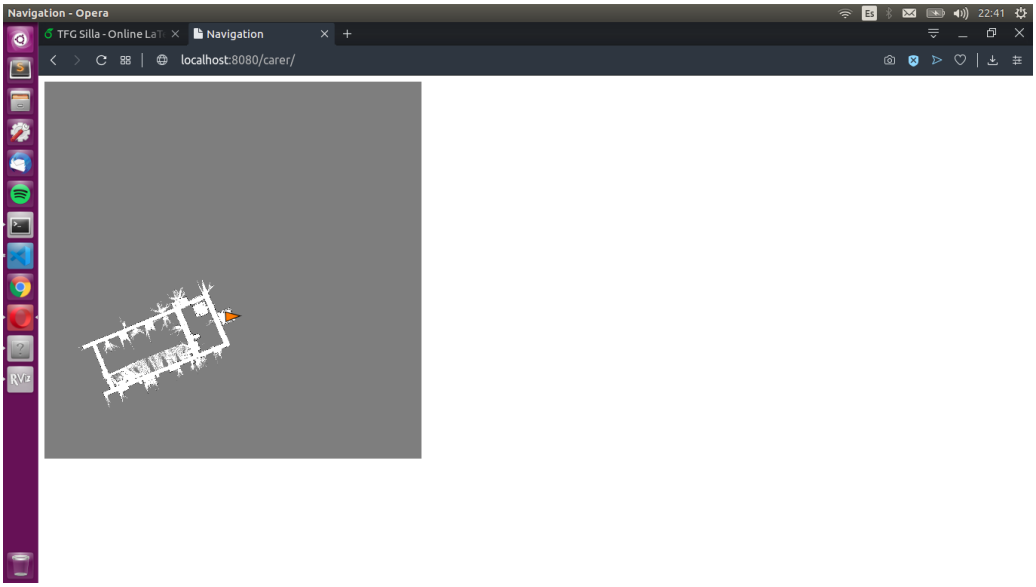


Figura 3.5: Mapa con el marcador de la silla

Dibujando el camino planificado

ROS2DJS también nos facilita el dibujo del camino que va a seguir nuestra silla. Para ello, se instancia un objeto de la clase `PathShape` y llama a la función `setPath(path)` que tiene como parámetro un mensaje del tipo `nav_msgs/Path`. Por último añadimos el objeto `PathShape` al canvas.

```
this.path = new ROS2D.PathShape({
    strokeColor: createjs.Graphics.getRGB(239, 48, 14, 0.66)
});

this.pathTopic.subscribe(function(message) {
    this.path.setPath(message);
});

this.gridClient.rootObject.addChild(this.path);
```

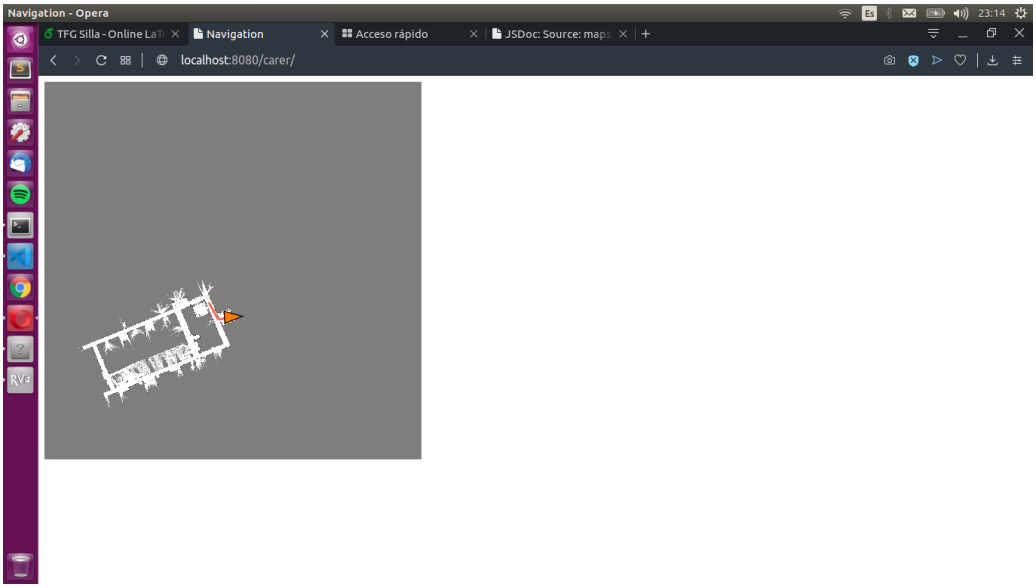


Figura 3.6: Mapa con el marcador del camino que seguirá la silla

3.1.3. Publicación de tópicos

Con la publicación de tópicos conseguimos mandar órdenes a la silla. Usamos el tópico `/move_base_simple/goal` para mandar a la silla a una posición y el tópico `/move_base/cancel` para detener el movimiento de la silla.

La publicación de un tópico es muy similar a cuando nos suscribimos a uno, simplemente esta vez usamos el método `publish(msg)` de la clase `Topic` que tiene como parámetro un mensaje de ROS.

Para crear un mensaje instanciamos un objeto de la clase `Message` pasándole al constructor un objeto JSON con sus campos.

```
set_goal(position){
    var goal_message = new ROSLIB.Message({
    header:
    {
        frame_id: "map"
    },
    pose:
    {
        position: position,
        orientation: {x:0, y:0, z:0, w: 1.0}
    }
    });

    this.goal = goal_message.pose
    this.goalTopic.publish(goal_message);
}
```

Al recibir este tópico el nodo correspondiente en nuestra silla, esta se moverá hasta las coordenadas indicadas. Lo mismo se haría con la publicación de un mensaje en el tópico `/move_base/cancel` para detener la silla.

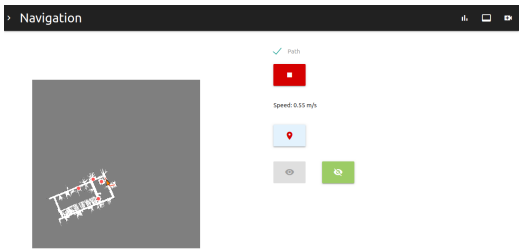


Figura 3.7: Movimiento de la silla al hacer doble click en el mapa

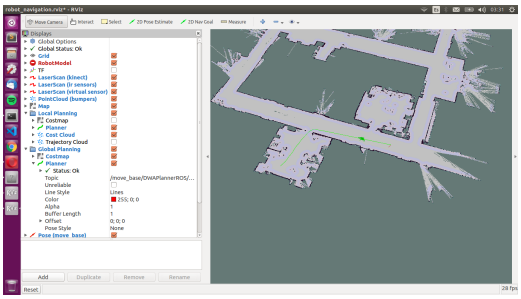


Figura 3.8: Simulador de la silla con la silla en movimiento

Aquí es posible encontrar más información sobre roslibjs y sobre ROS2DJS aquí. La documentación específica de ambas librerías se encuentra en documentacion ROSLIBJS y en documentacion ROS2DJS, respectivamente.

3.1.4. Visualización de tópicos imagen

Para la visualización de tópicos imagen usamos el paquete Web Video Server. Una vez instalado lo ejecutamos:

```
> rosparam set /web_video_server/port 8123
> rosrun web_video_server web_video_server
```

Haciendo una petición http a `http://localhost:8123/list_streams` obtenemos un array de streams disponibles:

```
{ "topics": [ "/camera/depth/image", "/camera/depth/image_raw", "/camera/rgb/image_raw", "/camera/rgb/image_color" ] }
```

Para visualizar uno de ellos simplemente usamos el elemento html `<img>` con la fuente /stream?topic=topico

```
<img src="/stream?topic=/camera/depth/image">
```

El nodo `web_video_server` se encuentra descrito en `http://wiki.ros.org/web_video_server`.

3.2. Integración con Janus

Una vez con Janus y su plugin Videoroom configurados y corriendo en nuestro servidor podemos interactuar con él usando su API JavaScript.

```
<img src="/stream?topic=/camera/depth/image">
```

En primer lugar debemos hacer es inicializar la librería mediante el método `Janus .init(options)`. Al que le podemos pasar como parámetro una función callback, que se ejecutará cuando termine el proceso.

```
Janus.init({
    debug: true,
    dependencies: Janus.useDefaultDependencies(),
    callback: function() {
        attach();
    }
});
```

Una vez hecho esto ya podemos usar el objeto Janus para crear una sesión con el servidor y agregar el plugin que queramos usar, en nuestro caso Videoroom. Una vez agregado el plugin, usando el atributo `success` podemos definir una función callback en la que se nos devolverá el *handle* del plugin. Este handle nos va a permitir crear publicadores y subscriptores de streams sobre el servidor de Janus.

```
function attach(){
    janus = new Janus(
    {
        server: '/janus',
        success: function() {
            janus.attach(
            {
                plugin: "janus.plugin.videoroom",
                success: function(pluginHandle) {
                    }});
                },
        }
    });
}
```

Más información sobre Janus disponible en la página de janus.

3.2.1. Plugin Videoroom

Lo primero que debemos hacer es crear una sala en videoroom, para ello se puede usar la API del plugin (petición 'create') o usar el fichero de configuración del plugin (janus.plugin.videoroom.jcfg).

Creando un *Publisher*

Para poder publicar en la habitación que hemos creado simplemente usamos una petición `join` con el *handle* que hemos recibido, con el atributo `ptype` igual a 'publisher' y el atributo `room` con la sala a la que nos queremos unir . Si la petición se hace con éxito, se nos enviará un mensaje de respuesta con el formato de la figura 3.9.

```
            success: function(pluginHandle) {
                videoroomPublisher = pluginHandle;
                videoroomPublisher.send({"message":{
                    "request" : "join",
                    "ptype" : "publisher",
                    "room" : 1234,
                }});
            },
```

```
{
        "videoroom" : "joined",
        "room" : <room ID>,
        "description" : <description of the room, if available>,
        "id" : <unique ID of the participant>,
        "private_id" : <a different unique ID associated to the participant; meant to be private>,
        "publishers" : [
                {
                        "id" : <unique ID of active publisher #1>,
                        "display" : "<display name of active publisher #1, if any>",
                        "audio_codec" : "<audio codec used by active publisher #1, if any>",
                        "video_codec" : "<video codec used by active publisher #1, if any>",
                        "simulcast" : "<true if the publisher uses simulcast (VP8 and H.264 only)>",
                        "talking" : <true|false, whether the publisher is talking or not (only if audio levels are used)>,
                },
                // Other active publishers
        ],
        "attendees" : [         // Only present when notify_joining is set to TRUE for rooms
                {
                        "id" : <unique ID of attendee #1>,
                        "display" : "<display name of attendee #1, if any>"
                },
                // Other attendees
        ]
}
```

Figura 3.9: Respuesta de éxito al unirse a una sala

Estas respuestas se recogen en la función callback definida en el atributo `onmessage` de las opciones de la función `janus.attach(options)`, que se llama al recibir un mensaje.

```
function attach(){
    janus = new Janus(
    {
        server: '/janus',
        success: function() {
            janus.attach(
            {
                plugin: "janus.plugin.videoroom",
                success: function(pluginHandle) {
                    videoroomPublisher = pluginHandle;
                    videoroomPublisher.send({"message":{
                        "request" : "join",
                        "ptype" : "publisher",
                        "room" : 1234,
                    }});
                },
                error: function(cause) {
                    console.log("Error en attach en publisher:", cause);
```

```
                },
                onmessage: function(msg, jsep){

                    if(event === "joined") {
                        joined = true;
                        $("#publicar").css("visibility", "visible");
                        $("#subscribir").css("visibility", "visible");

                        if(msg["publishers"] != undefined && msg["publishers"].length > 0){
                            participants = msg["publishers"].map(participant => participant.id)
                        }
                    }
                },
            })
        },
        error: function(cause) {
            console.log("Error al inicializar: ", cause);
        },
        destroyed: function() {
            console.log("Destroyed");
        }
    });
}
```

Una vez hecho esto, el usuario ya se habrá unido a la sala y podrá empezar a publicar, para ello debemos hacer una petición `publish`. Esta solicitud (12) debe ir acompañada de una oferta JSEP SDP para negociar una nueva conexión entre pares. El plugin responderá con una respuesta JSEP SDP, que debemos manejar usando `videoroomPublisher.handleRemoteJsep({jsep: jsep});`.

```
function publicar(){
    if(!publishing){
        videoroomPublisher.createOffer(
        {
            success: function(jsep) {
                var publish = { "request": "publish", "audio": true, "video": true };
                videoroomPublisher.send({"message": publish, "jsep": jsep});
            },
            error: function(cause){
                console.error("Error en publicar:", cause)
            }
        });
    }
}
```

Ahora el usuario estará publicando contenido en la sala y otros usuarios se podrán suscribir a él. El contenido que publiquemos lo podemos manejar en el callback especificado en el atributo `onlocalstream` de las opciones de la función `janus.attach(options)`

```
                onlocalstream: function(stream){
                    var video = document.querySelector("#local");
                    video.srcObject = stream;
                }
```

Creando un *Subscriber*

Los *subscribers* en videoroom (11) no son participantes sino simplemente *handles* que se usarán exclusivamente para recibir media de un *publisher* específico en la sala.

Por lo tanto debemos crear un nuevo handle del plugin y hacer especificar que se trata de un subscriber de determinada sala.

```
function addRemoteFeed(id){
    janus.attach(
    {
        plugin: "janus.plugin.videoroom",
```

```
        success: function(pluginHandle) {
            videoroomSubscriber = pluginHandle;
            videoroomSubscriber.send({"message":{
                "request" : "join",
                "ptype" : "subscriber",
                "room" : 1234,
                "feed" : id,
            }});
        },
    });
}
```

Esta petición (11) dará como resultado que el plugin prepare una nueva oferta de SDP que trate de negociar todas las transmisiones multimedia disponibles por el *publisher*. Para completar la configuración de PeerConnection, el suscriptor debe enviar una respuesta JSEP SDP al plugin. Esto se hace por medio de una solicitud de `start`, que debe estar asociada con una respuesta JSEP SDP usando `createAnswer`.

```
function addRemoteFeed(id){
    janus.attach(
    {
        plugin: "janus.plugin.videoroom",
        success: function(pluginHandle) {
            videoroomSubscriber = pluginHandle;
            videoroomSubscriber.send({"message":{
                "request" : "join",
                "ptype" : "subscriber",
                "room" : 1234,
                "feed" : id,
            }});
        },
        error: function(cause) {
            console.log("Error en attach subscriber:", cause);
        },
        onmessage: function(msg, jsep){
            var event = msg["videoroom"];

            if(event === "attached"){
                videoroomSubscriber.createAnswer(
                {
                    jsep: jsep,
                    media: { audioSend: false, videoSend: false },
                    success: function(jsep) {
                        var subscribe = { "request": "start" };
                        videoroomSubscriber.send({"message": subscribe, "jsep": jsep});
                    },
                    error: function(error) {
                    console.log("ERROR",error);
                    }
                });
            }
            else if(event === "event"){
                if(msg["started"]==="ok"){
                    listening = true;
                }
            }
            if(jsep !== undefined && jsep !== null) {
                videoroomSubscriber.handleRemoteJsep({jsep: jsep});
            }
        },
        onremotestream: function(stream){
            var video = document.querySelector("#remote");
            video.srcObject = stream;
        }
    });
}
```

En el callback del atributo `onremotestream` de las opciones de la función `janus.attach(options)` manejamos Los stream recibidos.

Más información sobre el plugin de Videoroom en `https://janus.conf.meetecho.com/docs/videoroom.html`.

3.3. Desarrollo de la interfaz de usuario

En este apartado describiremos el desarrollo de la interfaz de usuario, tanto de la parte del usuario de la silla como la parte del cuidador. Para este desarrollo hemos usado las tecnologías básicas de la web (HTML5, javascript, CSS), así como la librería jQuery, Google Charts y el framework Materialize.

3.3.1. Interfaz del cuidador

Figura 3.10: Elementos de la interfaz web del cuidador.

En la figura 3.10 se detallan las funciones de la interfaz del cuidador. Se puede apreciar que la parte principal de la interfaz es la visualización del mapa. La implementación de este mapa ya se ha explicado en la sección 3.1.1 Visualización del mapa. Se detallarán en esta sección todos los demás elementos que encuentran en la figura 3.10.

Estilos y componentes.

Para la parte visual hemos usado el framework Materialize. Con ello podemos usar sus componentes así como su `Grid` para hacer la página responsiva. Para la barra de navegación usamos su componente `NavBar` y para los botones usamos sus botones de tamaño grande con la clase `btn-large`.

Botones de cuidar y dejar de cuidar

Estos botones se implementan a modo de testigo. Suponiendo el caso de que haya varios cuidadores para un solo usuario de la silla, el botón de cuidar solo estará disponible si no hay alguna sesión activa que haya pulsado este botón. Una vez se pulse este botón el usuario cuidador tendrá el testigo, esto es, podrá usar las funciones de mover y parar la silla o la videoconferencia. Los demás cuidadores sin el testigo sólo podrán visualizar el mapa o visualizar las imágenes de la silla (tópico imagen).

Para implementar este testigo hacemos uso de la librería Socket.io en node. Cuando un usuario carga la página se crea una conexión con el servidor usando `io.connect()`. En el servidor cuando ocurra el evento de un usuario conectándose, se emite a este usuario el evento `available` si no hay un cuidador ocupándose. Así el usuario tendrá disponible el botón de cuidar.

```
carers_sockets.on('connection', function(socket){
    if(carer_id === null){
        socket.emit('available');
    }
}
```

Cuando un usuario pulse botón de cuidar, se emitirá un evento `occupied` que recibirán todos los cuidadores activos, deshabilitando el botón cuidar.

Cuando un usuario pulse botón de cuidar o se desconecte se le enviará a los demás usuarios activos el evento `available` de nuevo.

Eventos en el mapa

Para mover la silla a un lugar del mapa se debe hacer doble click en el lugar deseado. Lo que hacemos es escuchar el evento `dbclick` y recoger sus coordenadas para luego publicar (sección 3.1.3) un mensaje del tópico `/move_base_simple/goal` con esas coordenadas.

Podemos mover el mapa arrastrándolo con el ratón. Para ello usamos en combinación los eventos del ratón `mousedown` y `mousemove`. Usando la función `shift()` de la clase `Viewer` de ROS2DJS para mover el mapa en el canvas.

```
$( "#two-map" ).on("mousedown", function(event) {
    if(map_navigation.get_scene().mouseInBounds === true && !place_mode){
        $( "#two-map" ).on("mousemove", function(event) {
            map_navigation.shift_map(-event.originalEvent.movementX,event.originalEvent.movementY);
        });
    }
});
```

```
shift_map(x,y){
    this.viewer2D.shift(x,y);
}
```

Por otro lado, para aumentar o reducir el zoom usamos el evento del ratón `wheel` y el método `scaleToDimensions` de la clase `Viewer` de ROS2DJS.

```
$( "#twod-map" ).on("wheel", function(event) {
    if (map_navigation.viewer2D.scene.mouseInBounds === true) {
        event.preventDefault();
        map_navigation.zoom(event.originalEvent.deltaY);
    }
});
```

```
zoom(zoom){
    this.total_zoom = this.total_zoom + zoom;
    this.viewer2D.scaleToDimensions(this.scale_width + this.total_zoom, this.scale_heigth + this.total_zoom);
}
```

Además, podemos crear lugares de interés para el usuario de la silla. Es decir lugares del mapa definidos (por ejemplo la cocina o el baño) para que el usuario tenga botones en su interfaz que al pulsarlos lleven a la silla directamente hacia ese lugar. Para ello debemos pulsar el botón de modo para crear lugares de interés, una vez activado al hacer click en un lugar del mapa nos aparecerá un popup para ponerle un identificador a ese lugar. Este lugar con sus coordenadas lo guardaremos en nuestra base de datos implementada en SQLite.

Gráficas

Para dibujar los tópicos en gráficas usamos la biblioteca Google Charts. Lo primero que tenemos que hacer es cargarla.

```
//CARGAR LIBRERIA
google.charts.load('current', {packages: ['corechart']});
google.charts.setOnLoadCallback(drawChart);
```

Para dibujar una gráfica con Google Charts se ha definir una tabla. En el caso de las gráficas de linea de las que cogerá los valores de la coordenada del eje X de la primera columna y la coordenada Y de las columnas posteriores.

```
//DEFINIR TABLA
data = new google.visualization.DataTable();
data.addColumn('string', 'Element');
data.addColumn('number', 'latidos');
```

Lo siguiente es instanciar la gráfica usando el método del tipo de gráfica que queremos usar y pasándole en que elemento deseamos dibujarla. Y la dibujamos con la función `draw()`.

```
//INSTANCIAR Y DIBUJAR GRAFICA.
chart = new google.visualization.LineChart(document.getElementById('heartbeat_chart'));
chart.draw(data, options);
```

Por último, para dibujar la gráfica de un tópico en tiempo real, nos suscribimos a el e insertamos los datos que nos llegan en la tabla con el método `addRows`.

```
//INSERTAR FILAS A LA TABLA SEGUN NOS LLEGUEN LOS DATOS Y DIBUJAR DE NUEVO LA GRAFICA
heartbeat.subscribe(function(msg){
    if(count === 1){
        if(data. getNumberOfRows() < 40){
            data.addRows([['',msg.data]]);
```

```
        }
        else {
            data.removeRow(0);
            data.addRows([['',msg.data]]);
        }
        chart.draw(data,options);
        count = 0;
    }
    else {
        count++;
    }
}.bind(this));
```

Iremos representado los últimos 100 valores mostrando los datos en forma de ventana deslizante a lo largo del tiempo. En nuestro caso, hemos decidido omitir algunos datos que nos llegan, para cargar excesivamente el hilo principal. Además los datos que hemos usados han sido mockeados. El objetivo de este apartado es mostrar como se podrían representar los datos proporcionados por distintos sensores asociados a la silla como tópicos ROS. Además estos sensores mockeados se podrán sustituir por los sensores reales una vez que estén montados en la silla sin modificar la implementación.

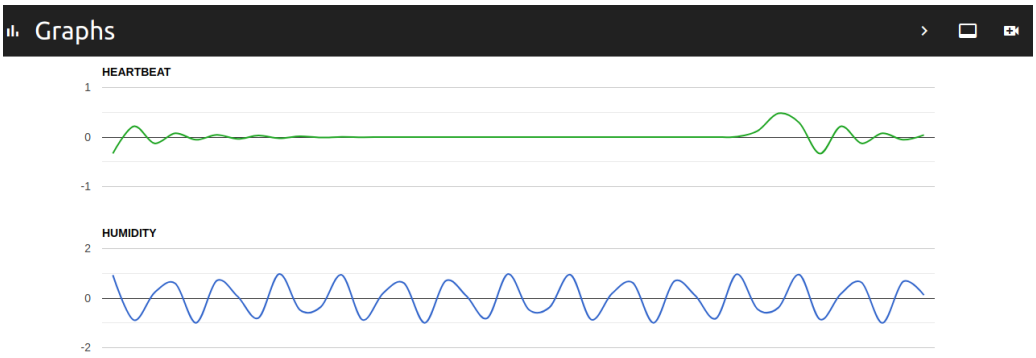


Figura 3.11: Gráficas de algunos tópicos.

Videoconferencia

El botón de videoconferencia abre una ventana emergente (figura 3.12) donde se ejecuta un script que crea un suscriptor y un publicador en Videoroom como hemos explicado en el apartado 3.2.1 Plugin Videoroom. Este modo permite visualizar tanto la imagen del usuario de la silla (en grande), como la del cuidador (en pequeño), e interactuar usando tanto audio como vídeo en con baja latencia, gracias al uso de comunicación punto a punto basada en WebRTC

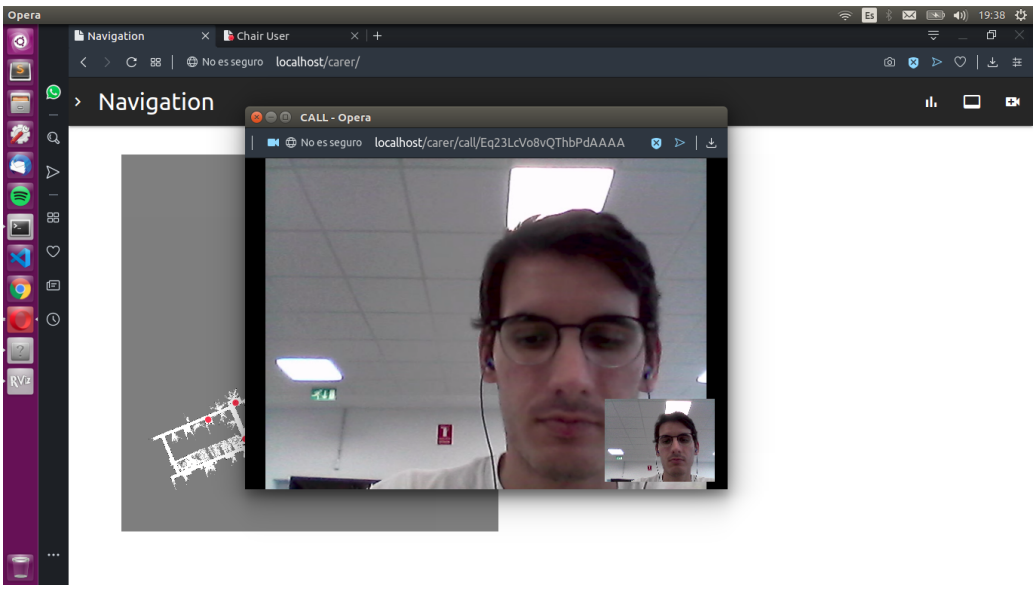


Figura 3.12: Videoconferencia cuidador.

3.3.2. Interfaz del usuario de la silla

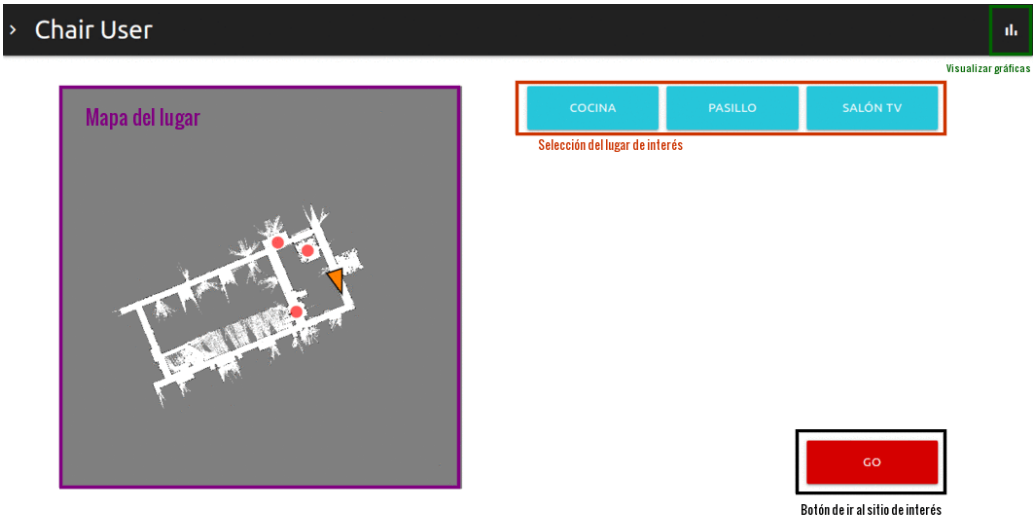


Figura 3.13: Elementos de la interfaz web del usuario de la silla.

Como se puede ver en la figura 3.13 la interfaz del usuario de la silla es muy similar a la interfaz del cuidador. Siendo los elementos diferenciadores que, por un lado la videoconferencia no se hace en una ventana emergente sino en la misma página, ocultándose el mapa (figura) y por otro lado es la visualización de los de lugares de interés, que el usuario puede seleccionar y luego pulsar el botón GO para ir a ese lugar. Ya que al pulsar este botón se publica (sección 3.1.3) un mensaje en el tópico `/move_base_simple/goal` con las coordenadas que cogemos de la base de datos SQLite. El grid de botones se ha diseñado de tal forma que pueda ser usado por distintos tipos de usuario. La silla viene equipada con una interfaz de control del

ratón basada en el seguimiento del movimiento del iris. Por este motivo, se han dispuesto distintos botones de gran tamaño en la mencionada rejilla, con un salto bien diferenciado entre botones. Además, el motivo del botón "GO"no es otro que el de evitar que el usuario dispare el movimiento de la silla de forma accidental, teniendo que pulsar tanto un objetivo como el mencionado botón.

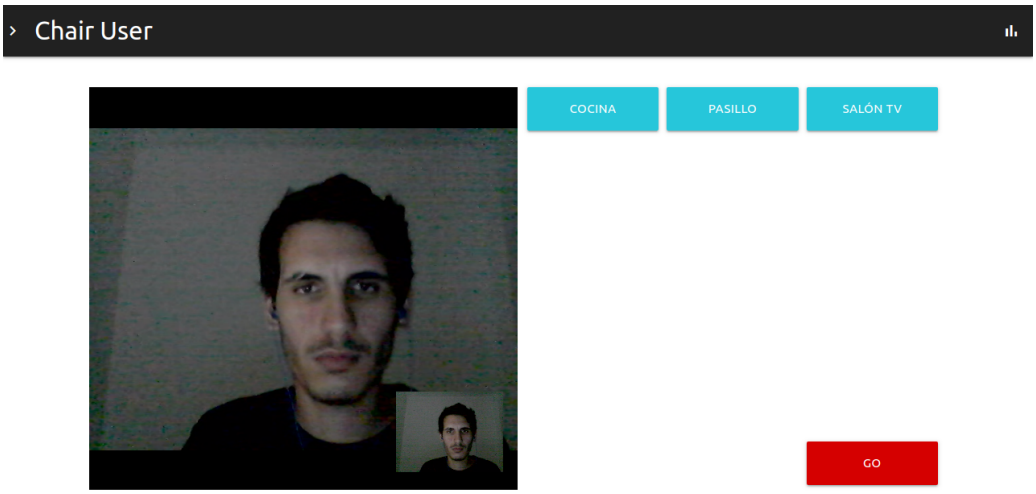


Figura 3.14: Videoconferencia en la interfaz de la silla.

Capítulo 4

Ejecución del prototipo

En este capítulo explicaremos como configurar el entorno para ejecutar poder ejecutar nuestro prototipo.

4.1. Configuración del entorno

4.1.1. Rosbridge

Suponiendo que ya tenemos nuestra silla con una versión kinetic de ROS funcionando. Lo primero que necesitamos instalar el nodo rosbridge para ello (5). Recordemos que este nodo nos permite la conversión entre tópicos de ros y mensajes cliente-servidor basados en websockets. En Ubuntu, ejecutamos el siguiente comando:

```
> sudo apt-get install ros-kinetic-rosbridge-server
```

4.1.2. Janus

El siguiente paso es instalar y configurar Janus. Lo primero es (3) instalar algunas dependencias que van a añadir soporte a diferentes códecs de vídeo y audio, soporte http/https, etc. En Ubuntu:

```
> aptitude install libmicrohttpd-dev libjansson-dev \
        libssl-dev libsrtp-dev libsofia-sip-ua-dev libglib2.0-dev \
        libopus-dev libogg-dev libcurl4-openssl-dev liblua5.3-dev \
        libconfig-dev pkg-config gengetopt libtool automake
```

Una vez instaladas las dependencias (3) descargamos el código y compilamos:

```
> git clone https://github.com/meetecho/janus-gateway.git
> cd janus-gateway
> sh autogen.sh
> ./configure --prefix=/opt/janus
> make
> make install
```

Janus requiere de distintos ficheros de configuración, que se encargan tanto de configurar el core como de los distintos componentes y plugins,

como el soporte para http o el propio videoroom plugin descrito anteriormente. La siguiente utilidad genera unos ficheros de configuración de base con los que empezar a trabajar:

```
> make configs
```

Hacemos algunas configuraciones en el fichero Janus.jfcg generado.

Configuramos la IP pública del servidor:

```
interface = "127.0.0.1"
```

Configuramos el servidor STUN , necesario para resolver los problemas derivados del NAT.

```
stun_server = "stun.l.google.com"
stun_port = 19302
```

Por otra parte, en el fichero janus.transport.http.jcfg especificamos que usaremos http y el puerto 8088:

```
http = false
port = 8088
```

Finalmente, en el fichero janus.plugin.videoroom.jcfg configuramos una sala:

```
room-1234: {
    description = "Demo Room"
    secret = "adminpwd"
    publishers = 20
    bitrate = 512000
    fir_freq = 10
    #audiocodec = "pcmu"
    videocodec = "vp9"
    #transport_wide_cc_ext = true
    record = false
    #rec_dir = "/path/to/recordings-folder"
    notify_joining = true
}
```

4.1.3. Web Video Server

Como ya vimos, el web video server se encarga de traducir tópicos de imagen de ros en imágenes que van pueden ser visualizadas directamente en un navegador web. Para su intalacion vamos a la carpeta de código de nuestro workspace.

```
> roscd && cd ../src
```

Descargamos el módulo correspondiente a `web_video_server`.

```
> git clone https://github.com/RobotWebTools/web_video_server.git
> git clone https://github.com/GT-RAIL/async_web_server_cpp
```

Compilamos

```
> roscd && cd .. && catkin_make
```

4.1.4. Nginx

Instalamos Nginx para que nos haga de servidor proxy inverso (4) :

```
> sudo yum install epel-release
> sudo yum update
> sudo yum install nginx
```

Añadimos lo siguiente en archivo de configuración de Nginx, (1) para configurar un servidor HTTPS, el parámetro ssl debe estar habilitado en los sockets de escucha en el bloque del servidor, y las ubicaciones del certificado del servidor y los archivos de clave privada deben especificarse:

```
server {
    listen 443 ssl;
    server_name *.tfg;
    ssl_certificate /etc/ssl/certs/nginx-selfsigned.crt;
    ssl_certificate_key /etc/ssl/private/nginx-selfsigned.key;
    ssl on;
    location / {
        proxy_pass http://127.0.0.1:8080;
    }
    location /janus {
        proxy_pass http://127.0.0.1:8088/janus;
    }
    location /list_streams {
        proxy_pass http://localhost:8123/list_streams;
    }
    location /stream {
        proxy_pass http://localhost:8123/stream;
    }
}
```

Con la directiva `location` redirigimos las peticiones a las rutan internas de los distintos servicios que tenemos corriendo.

4.1.5. Aplicación web

Descargamos el proyecto del repositorio de Github.

```
git clone https://github.com/alu0100904406/TFG_silla.git
```

El backend de la aplicación web ha sido desarrollado principalmente en node. Por lo tanto, necesitaremos instalar las dependencias necesarias antes de poder lanzar la aplicación.

```
npm install
```

4.2. Lanzamiento

Para lanzar los distintos servicios del backend, ejecutamos en distintas terminales los siguientes comandos:

`ros_bridge`.

```
> roslaunch rosbridge_server rosbridge_websocket.launch
```

`web_video_server`.

```
> rosparam set /web_video_server/port 8123
> rosrun web_video_server web_video_server
```

`Janus`.

```
> sudo /opt/janus/bin/janus -F /opt/janus/etc/janus/
```

`node`. En la carpeta del proyecto.

```
> node server
```

Solo tendríamos que cargar la página /carer o /chair en el navegador.

4.3. Manual del usuario

En esta sección se explican las distintas funciones de la aplicación desde la perspectiva del usuario, y su forma de uso.

4.3.1. Manual del cuidador

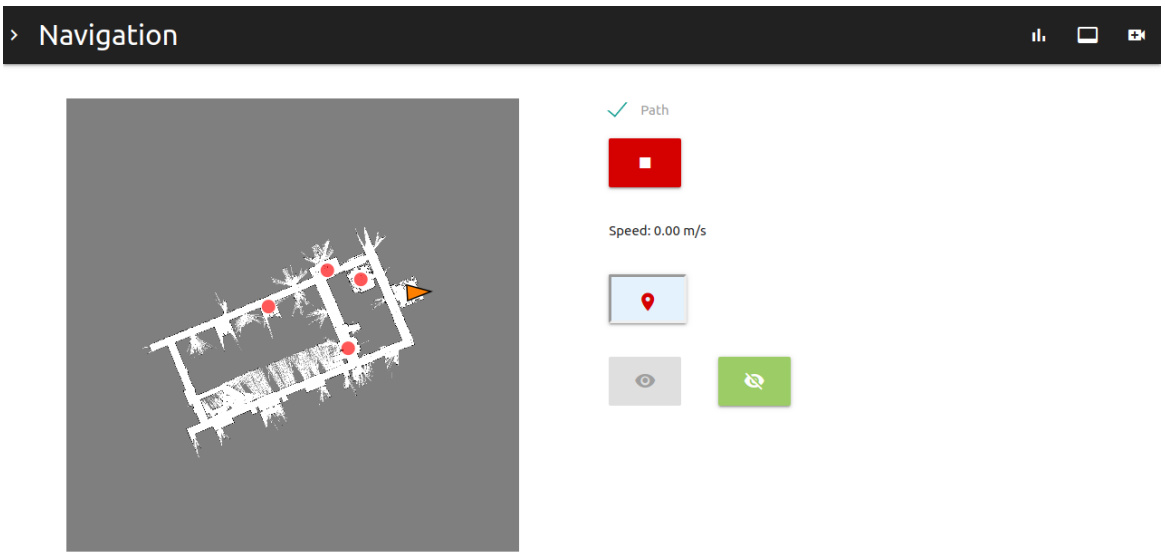


Figura 4.1: Interfaz web del cuidador.

Para poder hacer uso de las distintas funciones del panel de control del cuidador debe pulsar el botón con el icono 👁. Si este botón no le aparece disponible significa que ya hay otro cuidador activo. Una vez usted esté como cuidador activo tendrá acceso a las siguiente funciones:

- **Mover la silla a un lugar del mapa.** Para mover la silla a un lugar del mapa, hacer doble click sobre dicho lugar del mapa.
- **Zoom del mapa.** Para aumentar o reducir el tamaño del mapa usar la rueda del ratón mientras el puntero está situado sobre el mapa.
- **Zoom del mapa.** Para mover el mapa basta con arrastrarlo con el ratón.

- **Puntos de interés** Para añadir un marcador de punto de interés en el mapa debe pulsar el botón con el icono . Aparecerá un modal para introducir el identificador de dicho lugar. Para eliminar o modificar un punto de interés haga click sobre su marcador y aparecerá un modal donde podrá eliminarlo o modificar su nombre.
- **Parar silla.** Para cancelar el movimiento de la silla pulse el botón con el icono .
- **Visualizar graficas.** Para visualizar gráficas relacionadas con la silla pulse el botón de la barra de navegación .
- **Visualizar imágenes de la cámara de la silla.** Para visualizar una transmisión de imágenes de la silla pulse el botón de la barra de navegación.
- **Videollamada.** Para realizar una videollamada pulse el botón de la barra de navegación. Debe permitir el uso de su cámara y micro.

4.3.2. Manual del usuario de la silla

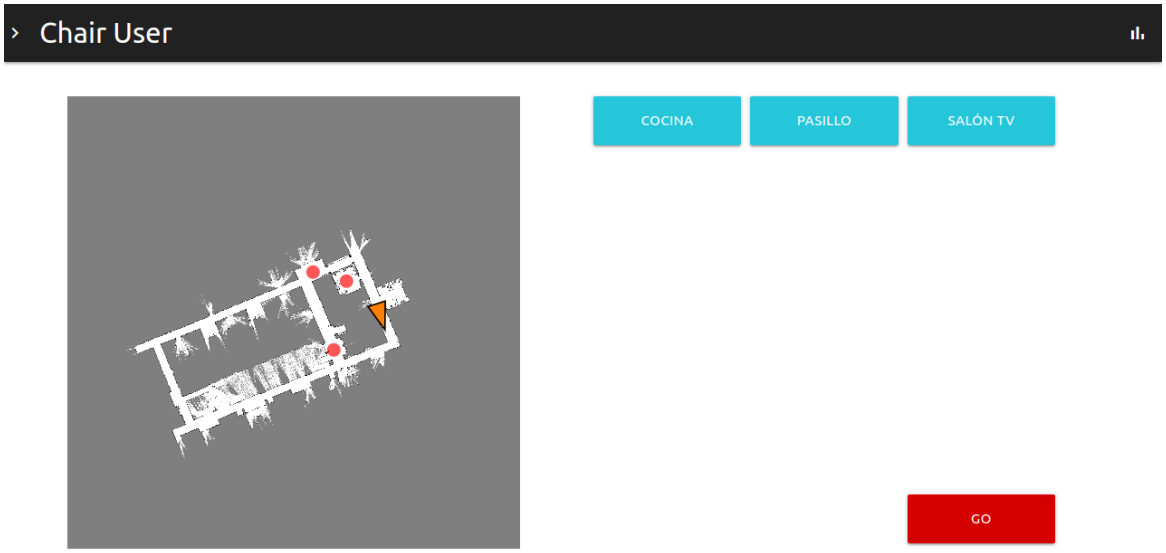


Figura 4.2: Interfaz del usuario de la silla.

- **Mover la silla a un lugar de interés.** Para mover la silla a un lugar del mapa, hacer doble click sobre dicho lugar del mapa.
- **Zoom del mapa.** Para aumentar o reducir el tamaño del mapa usar la rueda del ratón mientras el puntero esta situado sobre el mapa.

- **Zoom del mapa.** Para mover el mapa basta con arrástralo con el ratón.
- **Visualizar gráficas.** Para visualizar gráficas relacionadas con la silla pulse el botón de la barra de navegación .
- **Videollamada.** Al recibir una llamada debe permitir el uso de su cámara y micro.

Capítulo 5

Conclusiones y líneas futuras

Como conclusión cabe destacar que la aplicación es completamente funcional y cumple con los objetivos que se fijaron. El proyecto tiene una estructura modular que permite la adición de nuevas funcionalidades de forma sencilla, o incluso incorporar estas funcionalidades a otros proyectos. El proyecto, además hace uso de muchas tecnologías web que se están usando en la actualidad, pudiendo ser esta memoria de ayuda para un usuario que quiera introducirse en, por ejemplo, tecnologías WebRTC, ROS u otras aquí descritas. La interfaz de usuario tiene en cuenta las potenciales limitaciones de los usuarios y sus interfaces con la máquina, y permite la personalización por parte del cuidador a la hora de visualizar la información que más le interesa en cada momento.

Como lineas futuras se proponen las siguientes mejoras:

- Mejoras en la seguridad, añadiendo una caducidad de la sesión o un autenticación.
- Si bien el diseño ya es modular, esta modularidad se podría mejorar permitiendo añadir una especie de framework para añadir botones que publiquen ciertos tópicos o visualizaciones de otros tópicos.
- Mejoras en la accesibilidad, hacer que la página cumpla con los criterios WCAG.
- Sustitución del módulo ROS2DJS por uno más eficiente, mediante un desarrollo propio de cero o mediante la adaptación de la librería ya existente.
- Añadir distintos tipos de sensores, como LIDARs o ultrasonidos.

Capítulo 6

Summary and Conclusions

In this final project we developed a web interface that allows controlling an autonomous wheelchair. An autonomous wheelchair is a chair capable of moving towards a target location in an interior or exterior area.

In addition, we have implemented tools to monitor the chair and its user, and a module that allows communication between the user of the chair and a potential caregiver through videoconference.

The application is fully functional and meets the objectives that were set. The project has a modular structure that allows the addition of new functionalities in a simple way, or even incorporate these functionalities to other projects. The project also makes use of many web technologies that are currently being used, so this document could be helpful for users who wants to introduce themself in some of the technologies described in this document.

Capítulo 7

Presupuesto

Este capítulo recoge un posible presupuesto aproximado de la realización del proyecto. El ámbito de este proyecto se limita al desarrollo de una aplicación software que permita el control de una silla robotizada, la cual se presupone terminada y funcional. Por lo tanto, los costes asociados a la misma se asumen externos al presente proyecto y no se tendrán en cuenta a la hora de calcular un presupuesto.

Descripción	**Coste**
Tiempo del alumno dedicado al desarrollo	3200€
PC que actúe como servidor en la silla	700€
PC para el lado del cuidador	300€
Cámara web	30€

Tabla 7.1: Presupuesto.

Como se puede apreciar en el cuadro 7.1 el presupuesto total sería de 4120€

Bibliografía

[1] Configuring https servers. http://nginx.org/en/docs/http/configuring_https_servers.html.

[2] Express - infraestructura de aplicaciones web node.js. https://expressjs.com/es/.

[3] meetecho/janus-gateway: Janus webrtc server. https://github.com/meetecho/janus-gateway.

[4] Nginx docs | installing nginx open source. https://docs.nginx.com/nginx/admin-guide/installing-nginx/installing-nginx-open-source/.

[5] rosbridge_suite. http://wiki.ros.org/rosbridge_suite.

[6] Ros.org | powering the world's robots. https://www.ros.org.

[7] Socket.io. https://socket.io.

[8] Una silla de ruedas autónoma que se conduce sola - tecnobility. tecnología, discapacidad y mayores. https://www.tecnobility.com/es/noticia/una-silla-de-ruedas-autonoma-que-se-conduce-sola.

[9] What is nginx? https://www.nginx.com/resources/glossary/nginx/.

[10] Janus webrtc server: About janus. https://janus.conf.meetecho.com, 2014.

[11] Videoroom plugin documentation. https://janus.conf.meetecho.com/docs/videoroom, 2019.

[12] Videoroom plugin documentation. https://janus.conf.meetecho.com/docs/videoroom.html, 2019.

[13] de La Laguna, U. Project perenquén - verdino research project. http://verdino.webs.ull.es/project-perenquen.html, 2015.

[14] Dutton, S. https://www.html5rocks.com/en/tutorials/webrtc/infrastructure/. https://www.html5rocks.com/en/tutorials/webrtc/infrastructure/, 2013.

[15] Huang, J. Ros tutorial 2: Publishers and subscribers. https://www.youtube.com/watch?v=bJB9tv4ThV4.

[16] Saito, I. web video server - ros wiki. http://wiki.ros.org/web_video_server, 2011.

[17] Toris, R. ros2djs - ros wiki. http://wiki.ros.org/ros2djs, 2015.

[18] Uberti, J., and Dutton, S. Real-time communication with webrtc: Google i/o 2013. https://www.youtube.com/watch?v=p2HzZkd2A40.