UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Politécnica Superior de Gandia

JinkoBot: ROBOT terapéutico para niños con trastorno del espectro autista (TEA)

Trabajo Fin de Grado

Grado en Tecnologías Interactivas

AUTOR/A: Hernández Soler, Diana

Tutor/a: Pérez Pascual, Mª Asunción

CURSO ACADÉMICO: 2021/2022

Resumen

Jinkobot es un robot de asistencia en terapia con niños autistas. Se trata de usar la tecnología, en concreto la robótica, para favorecer la interacción del niño con los ejercicios que son utilizados habitualmente en este tipo de terapias.

El robot será el que guía al niño a través de los ejercicios que el psicólogo o terapeuta ha escogido previamente mediante una aplicación web.

Una vez realizados, el psicólogo podrá ver en la aplicación web el progreso del niño a lo largo de las sesiones en forma de gráficas dónde se podrá encontrar el tiempo empleado en cada respuesta y el número de aciertos.

De esta manera se pretende, no solo ayudar al niño durante la realización de sus ejercicios, sino también al profesional, recolectando y procesando datos objetivos de manera rápida y ordenada.

Palabras clave: Autismo, TEA, Robótica, ROS

Abstract

Jinkobot is a robot used to assist therapists in their work with autistic children. This project aims to use technology, specifically robotics, to promote the child's interaction with the exercises and routines that are generally used in this type of therapy.

The robot will guide the child through the exercises that the psychologist or therapist has previously prepared through a web application.

Once the exercises have been completed, the psychologist will be able to see the child's progress throughout the sessions, as graphs, in the web application.

These graphs will contain information such as the time spent on each answer, the number of correct answers, and the number of unanswered questions, amongst others.

The purpose of this project is to help the child, as well as the therapist, during the performance of the exercises by collecting and processing objective data that would otherwise be much more complex to monitor.

Keywords: Robotics, ASD, Autism, ROS

Índice

Listado de figuras

Listado de tablas

Capítulo 1

INTRODUCCIÓN

El trastorno del espectro autista, TEA en adelante, es un tipo de diversidad en el desarrollo que puede provocar dificultades sociales, comunicacionales y conductuales significativas.

Si algo caracteriza a las terapias para TEA es la alta interacción personal entre el profesional y el paciente. Mediante una serie de ejercicios, se puede conseguir una mejora constante en diferentes aspectos, como son las habilidades sociales, la flexibilidad cognitiva entre otros aspectos.

Otro factor para tener en cuenta es la elevada presencia de síntomas del trastorno por déficit de atención e hiperactividad (TDAH) en niños con un diagnóstico clínico de autismo. Esto dificulta mucho el desarrollo de los ejercicios, ya que el niño no es capaz de concentrar su atención durante el tiempo necesario para llevar el ejercicio a cabo.

En una publicación de la revista Espacios *[4]*, Fausto R. Cabrera describe como la atención de niños con TDAH mejora ante un elemento tecnológico, ya que éste les crea curiosidad. Se trata de una terapia con la participación directa de un robot humanoide, concretamente el robot NAO [Anexo 2], en la que se consiguió, claramente, un mayor grado de atención y aceptación por parte de niños con diferentes grados TDAH tal y com ose muestra en la tabla 1.

Nº de pregunta	**Aceptación**	**Rechazo**
1	100%	0%
2	95%	5%
3	100%	0%
4	100%	0%
5	90%	10%

Tabla 1:. Encuesta realizada a niños de la escuela "San Ignacio de Loyola" Fuente: [4]

En otro estudio coordinado por Tapus, A. en 2012 [1] se presentan una serie de experimentos para tratar de dotar de fundamentos la siguiente hipótesis: Los niños con TEA muestran un mayor grado

de interacción social en presencia del robot NAO. [1] En la gráfica de la Figura 1 se pueden observar las diferencias de interacción que se dan ante el robot NAO y ante un humano.

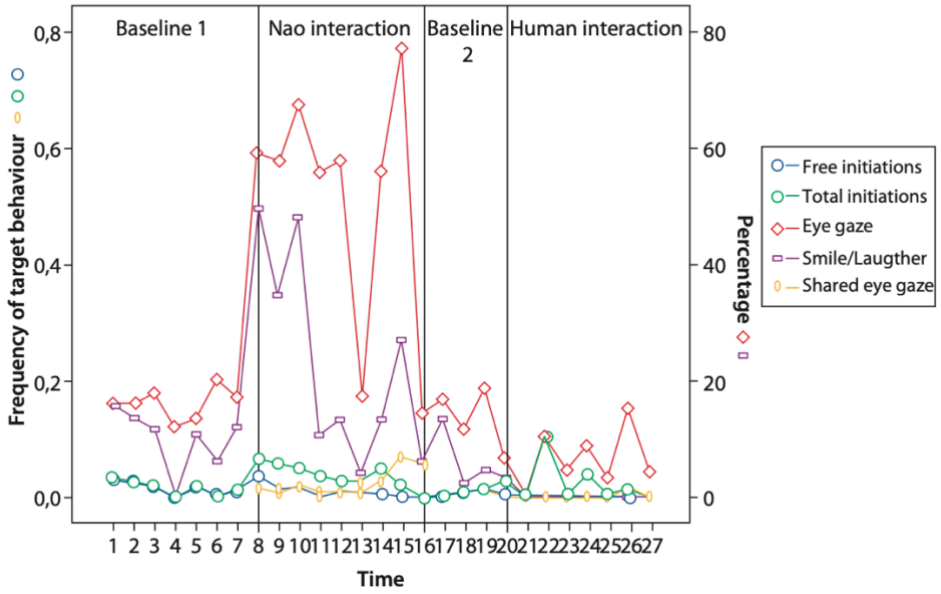


Figura 1: Nivel de interacciones con un robot y con un humano. Fuente: [1]

Si atendemos a la gráfica podemos observar como el contacto visual ante el robot destaca sobrela interacción con un humano. Lo mismo ocurre con la sonrisa y/o risa del niño ante el robot.
Esto nos indica lo favorable que sería la interacción con un robot para la realización de los ejercicios en terapia ya que el niño presentaría, en términos generales, una mayor predisposición a mantener contacto visual con el robot y también, un mejor humor que favorecería la atención en la realización de los ejercicios.

Cabe destacar que los trastornos del espectro autista presentan un alto rango de manifestaciones, por lo que en este trabajo nos hemos centrado en facilitar la realización de los ejercicios y las rutinas en las sesiones de terapia.

A continuación, se detalla la estructura de la memoria, que consta de los siguientes capítulos:

Capítulo 1: **Introducción**

En esta primera sección de la memoria se detalla una introducción al contexto del proyecto, así como la estructura del documento.

Capítulo 2: **Objetivos**

Detalle de ls objetivos de este proyecto, tanto el principal como los secundarios asociados al propio desarrollo. Se trata de un listado de lo que se pretende conseguir mediante este proyecto final de grado.

Capítulo 3: **Metodología**

En este apartado se especificará la metodología utilizada para el desarrollo de este proyecto. CDIO en este caso, a saber, concepción, desarrollo, implementación y operación.

Capítulo 4: **Diseño**

Toda la parte de diseño, tanto de la programación del robot mediante ROS como de la aplicación web con la librería *React* se describe en este apartado.

Capítulo 5: **Implementación**

Todo el proceso de implementación estará plasmado en este capítulo, desde el robot, la aplicación web, así como el servicio de base de datos en la nube.

Capítulo 7: **Testeo**

Uno de los aspectos más importantes en el desarrollo de cualquier tipo de software. Aquí se detalla unos ejemplos de testeo tanto de la aplicación web como también de la aplicación en ROS.

Capítulo 8: **Resultados**

En este apartado se detallarán los resultados obtenidos con este proyecto, un video hará de muestra de todo lo implementado.

Capítulo 9: **Conclusiones**

En este último apartado se reúnen las conclusiones de este proyecto final de grado. Una manera de hacer retrospectiva a través de todo el tiempo invertido en este proyecto.

Esta memoria terminará con un listado de la bibliografía utilizada a lo largo de todo este proyecto y también con anexos que complementan toda la información aportada, tales como las actas de reuniones con el psicólogo, un estudio de los robots para terapia con TEA en la actualidad etc.

Capítulo 2

OBJETIVOS

El objetivo principal del proyecto es acercar la tecnología y todo lo que con ella se puede conseguir al mundo terapéutico, existen muchos tipos de diversidad en el mundo, y la tecnología puede mitigar, o al menos intentar ayudar, en gran cantidad de ellos.

Como objetivos secundarios para este proyecto podemos enumerar los siguientes:

- Analizar proyectos similares de robots aplicados a personas con trastorno del espectro autista.
- Realizar una toma de requisitos lo más cercana a la realidad junto con un profesional especializado en terapia con niños con TEA.
- Diseñar e implementar una aplicación web para el control de los ejercicios terapéuticos a realizar con niños con TEA mediante un robot.
- Diseñar e implementar los paquetes necesarios para controlar las acciones del robot relacionadas con la aplicación del objetivo anterior.
- Diseñar e implementar un servicio de *backend* en la nube para gestionar tanto base de datos como el proceso de *login.*

Siempre he sentido la tecnología como una manera de ayudar a las personas, y este TFG me ha permitido llevar a cabo esta idea. Usar la tecnología en la terapia con niños con TEA, para ayudar al niño durante los ejercicios, pero también al psicólogo durante la evaluación de los resultados.

Capítulo 3

METODOLOGÍA

Para este proyecto se ha optado por la utilización de la metodología CDIO. Esta metodología está basada en el ciclo de vida del producto. A saber, concepción, diseño, implementación y operación. Fases que se muestran en la figura 2.

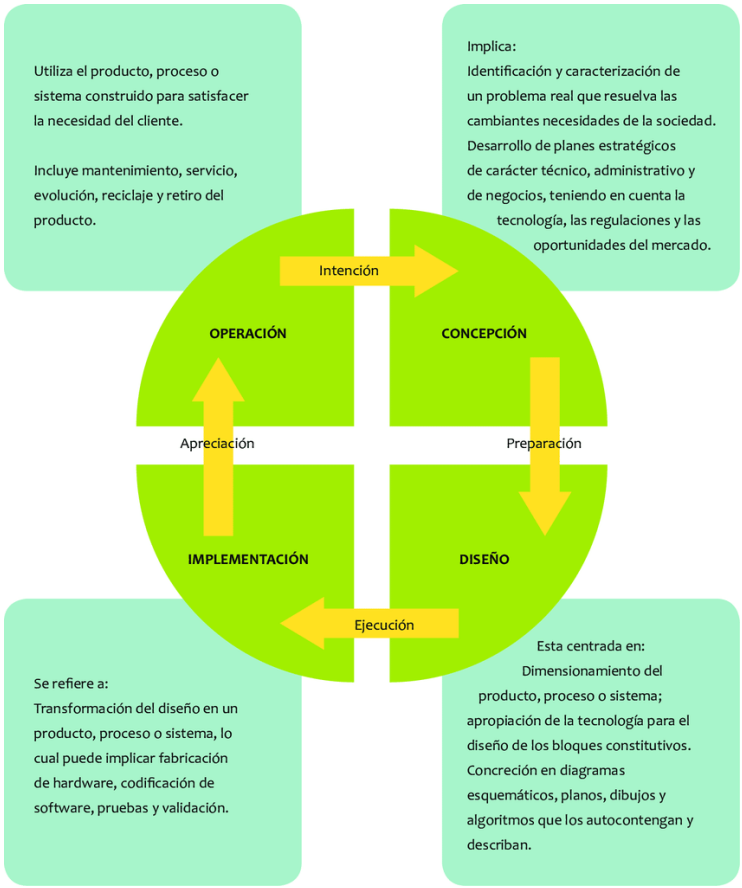


Figura 2: Fases de la metodología CDIO Fuente: [7]

Una de las razones para elegir esta metodología en concreto es su última fase, es decir, la operación, dónde se tiene en cuenta futuras ampliaciones, actualizaciones y uno de los aspectos más importantes para tener en cuenta en la actualidad debido a la vorágine tecnológica en la que vivimos: el reciclado. Qué hacer con el producto una vez su vida útil haya concluido.

También se ha optado por esta metodología como una despedida de este grado, cerrando el circulo y utilizando la metodología que utilizamos hace ya 4 años, en aquel ilusionante primero, para este último proyecto dentro de la universidad.

Concepción

La fase de concepción se ha dividido en dos partes, por un lado, toda la investigación previa sobre el tema, y por el otro, la ayuda de un profesional para delimitar bien los requerimientos reales de este proyecto.

Al tratarse de un tema completamente nuevo para mi comencé con la lectura de artículos e información relacionada con el autismo en niños, de los cuales me gustaría destacar lo siguiente:

- Los buenos resultados que se pueden observar mediante la interacción con un robot de niños con diferentes grados de TEA. En todos los estudios y artículos consultados se ha encontrado un mayor grado de atención por parte del niño en comparación con la interacción con una persona. [1, 2, 3]
- En uno de ellos [2] se concluye que no es necesario una apariencia, ya sea humana o animal, para conseguir esta atención por parte del niño.
- Algunos estudios sugieren que los niños con TEA se sienten mucho más cómodos en ambientes controlados y predecibles. [3]

Como método de investigación complementario se ha realizado un listado histórico de los robots enfocados en terapias de autismo [Anexo 8.2] así como como uno con los diferentes robots que se pueden encontrar en el mercado actual enfocados en la misma temática. [Anexo 8.3]

Aisoy	Leka	Robot social IO	RobTEA
			

Tabla 2: Robots para TEA en la actualidad

Como ya se ha comentado al principio también conté la ayuda de *Ramón Bonilla Calvo*, psicólogo especializado en niños con TEA, que ha sido mi guía en materia clínica a lo largo de todo este proyecto.

En varias reuniones [Anexo 8.1] con él pude hacer un esbozo de los aspectos más importantes a los que se prestaba atención durante una terapia de estas características, llegando a unas primeras conclusiones de las que se pueden destacar las siguientes:

- Los juegos y/o ejercicios son bastante mecánicos y repetitivos.

- Al interaccionar el niño con el robot el psicólogo tiene una mayor libertad en observar el comportamiento del niño con mayor detalle, no tiene que estar pendiente de si el resultado es correcto o no y tomar nota.

- Una vez terminados los ejercicios, resulta interesante poder guardar estadísticas de su ejecución, como, por ejemplo, cuanto tiempo ha tardado en responder y si la respuesta ha sido la correcta o no.

- Poder saber la media de acierto que tiene el ejercicio entre los usuarios. Para saber si el niño va progresando adecuadamente, como la mayoría, o hace falta reforzar algún área en concreto.

Tras estas conclusiones se pasó a analizar los diferentes ejercicios que se realizan en una sesión de terapia y se decidió la realización de dos tipos de ejercicios que engloban de alguna manera los ejercicios prototípicos en este tipo de terapias.

Reconocimiento de contextos y emociones mediante pictogramas

Una gran parte de los ejercicios que se utilizan para el tratamiento de niños con TEA están relacionados con las inferencias y emociones.
Los niños dentro de este espectro suelen presentar, en mayor o menor medida, incapacidad para el reconocimiento de expresiones faciales y su significado emocional lo que dificulta enormemente su interacción social.

Ejercicios de perspectiva y orientación

Con este tipo de ejercicios se pretende acostumbrar al niño a entornos donde deben seguir unas directrices, para evitar así conductas de fuga, así como fomentar el reconocimiento y atención mediante el seguimiento del robot.

Una aplicación web estaría en conexión con el robot tanto para su control como para poder visualizar las gráficas y estadísticas referentes a las rutinas y sesiones del paciente.
Una vez decidido la parte no técnica del proyecto, vamos a pasar a los requisitos técnicos.

Diseño

Para el desarrollo y la programación del robot se ha elegido ROS (*Robot operating system*). ROS es un set de paquetes y librerías para el desarrollo de aplicaciones en robótica.

Una de las principales razones para utilizar ROS es que es un proyecto *open source* (OS) que se puede visitar en su repositorio de *github*. El software open source es código accesible al público: se puede ver, modificar y distribuir. Se desarrolla de manera colaborativa, es decir, depende de la revisión de compañeros y de la producción de la comunidad. Suele ser más económico y flexible que sus alternativas propietarias, porque el desarrollo corre a cargo de las comunidades y no de una sola empresa o persona.

Una aplicación web con sistema de *login* mediante usuario y contraseña completa este proyecto.
Esta aplicación es el complemento perfecto para el terapeuta y/o psicólogo. Desde ella podrá gestionar y visualizar los datos obtenidos por el robot. Esto será posible gracias a la conexión con una base de datos en la nube. También estará conectada al robot para iniciar los ejercicios.

Implementación

Para la realización de este proyecto final Grado se han hecho uso de un gran abanico de tecnologías con la intención de realizar un producto que de verdad supliera una necesidad. No se trata solo de realizar una aplicación web, o programar un robot con ROS, si no de poner en práctica la capacidad adquirida durante estos cuatro años del grado para implementar un producto tecnológico que pudiera ayudar a las personas. Se detallan, a continuación, todas las tecnologías utilizadas en este proyecto:

ROS

ROS es un sistema operativo de código libre que se puede programar a través de Python o de C++, se ha elegido Python ya que cuenta con grandes librerías para el procesado de visión artificial, parte sumamente importante para este proyecto.

Librería OpenCV de Python

La librería elegida para el procesamiento de imágenes ha sido OpenCV, también de código abierto, que permite una gran variedad de acciones para procear la información recibida desde la cámara del robot.

React y Next.js

El desarrollo web de los últimos años está poniendo especial énfasis en la programación por componentes, llevando el "Divide y vencerás" a otro nivel.

Un nivel que permite un código más mantenible, los ficheros son infinitamente más pequeños, y totalmente reutilizable, lo que agiliza muchísimo el desarrollo de nuevas aplicaciones.

Esta es la razón por la que se ha optado por utilizar el framework Next.js que permite utilizar la librería React de una forma mucho más sencilla.

Firebase

Se trata de un servicio integral de infraestructura en la nube. Tiene muchas funcionalidades, aunque para este proyecto se ha hecho uso de la base de datos y del servicio de autentificación mediante usuario y contraseña.

Amazon Polly

Amazon Polly es un servicio que convierte texto en habla. *Polly* es un servicio de texto a voz (TTS) que utiliza tecnologías de aprendizaje profundo para que se asemeje a una voz humana.

Una de las características más interesantes de este servicio es que permite el ajuste el estilo de habla, la frecuencia, el tono y el volumen de la voz. Características que se ha utilizado, por ejemplo, en el caso de que el niño no responda correctamente la pregunta, ya que, en ese caso, el robot volverá a realizar la pregunta más lento y haciendo más énfasis en las palabras clave.

Operación

En esta fase hay que tener en cuenta dos aspectos fundamentales, al tratarse de software la necesidad de soporte y/o actualización tanto del robot como de la aplicación web.

Pero por otro lado también hay que tener presente que este proyecto cuenta con un robot con componentes perjudiciales para el medio ambiente, como las baterías, por ejemplo, y hay que tener una buena información medioambiental al respecto para una vez llegado al fin de su uso, ya sea por obsolescencia o por deterioro o avería poder saber cuál es el protocolo para seguir para minimizar el impacto en el medio ambiente.

Actualmente uno de los principales problemas es el desecho de este tipo de basura, porque la mayoría de estos aparatos contienen elementos tóxicos que, de no ser adecuadamente tratados, pueden afectar de manera muy nociva tanto al medio ambiente como a la salud de las personas.

EL RD 110/2015 recoge todas las etapas que van desde la puesta en el mercado de los AEE a la recogida y gestión como residuos destacando la peligrosidad de los componentes haciendo hincapié en la prevención de la generación de residuos. Valorando una posible reutilización de estos antes de proceder a su eliminación. [6]

Los principales aspectos relacionados con la gestión de los residuos de los AEE, desde su generación hasta su tratamiento específico son:

- La recogida separada. (Entes Locales, distribuidores y gestores).
- Obligaciones y responsabilidades de todos los agentes implicados en la recogida separada, así como el comportamiento medioambiental de los consumidores.
- La correcta aplicación de la responsabilidad ampliada del productor.
- Requisitos de las instalaciones de recogida.
- Condiciones de almacenamiento.
- Preparación para la reutilización, reciclado y tratamiento específico.

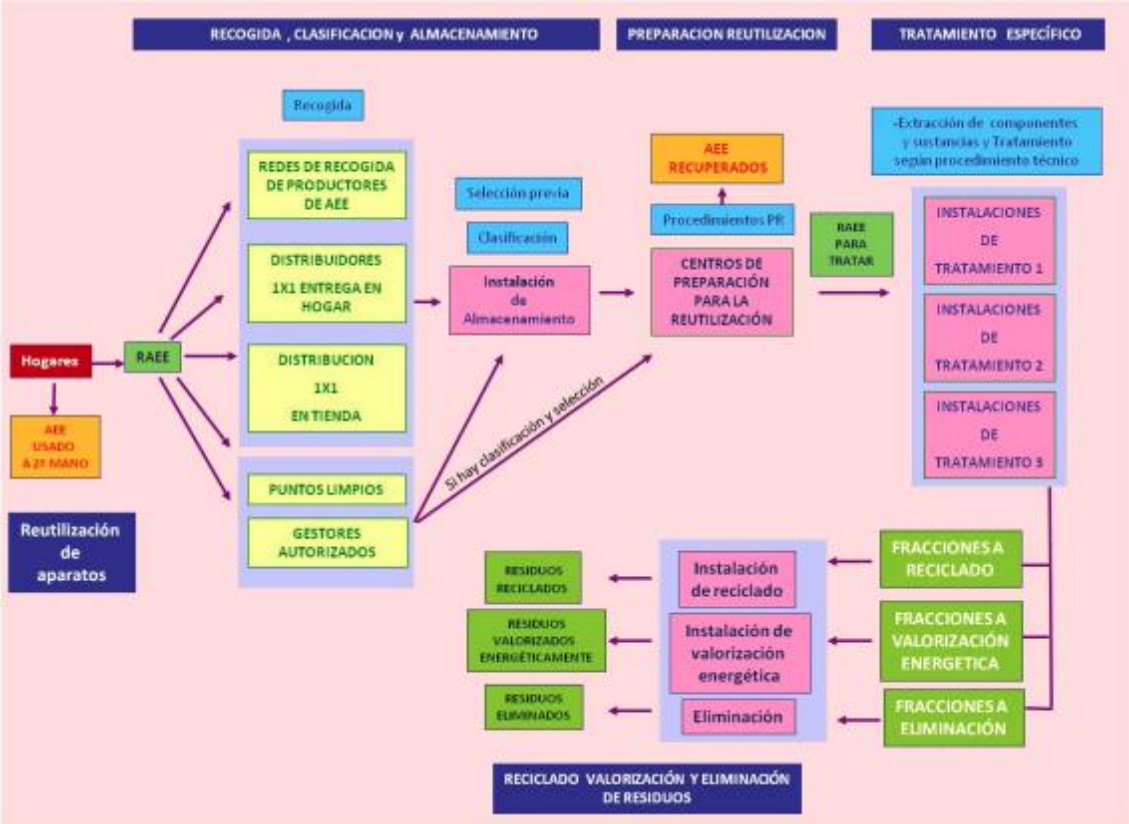


Figura 3: Esquema general de la gestión de RAEE. Fuente: [6]

Capítulo 4

DISEÑO

En este capítulo vamos a adentrarnos en el diseño de todo el proyecto, estableciendo historias de usuario y esquemas para entender mejor la futura implementación. Se detallarán también las pantallas y su distribución que tendrá nuestra aplicación web, así como todo lo relacionado con la programación del robot.

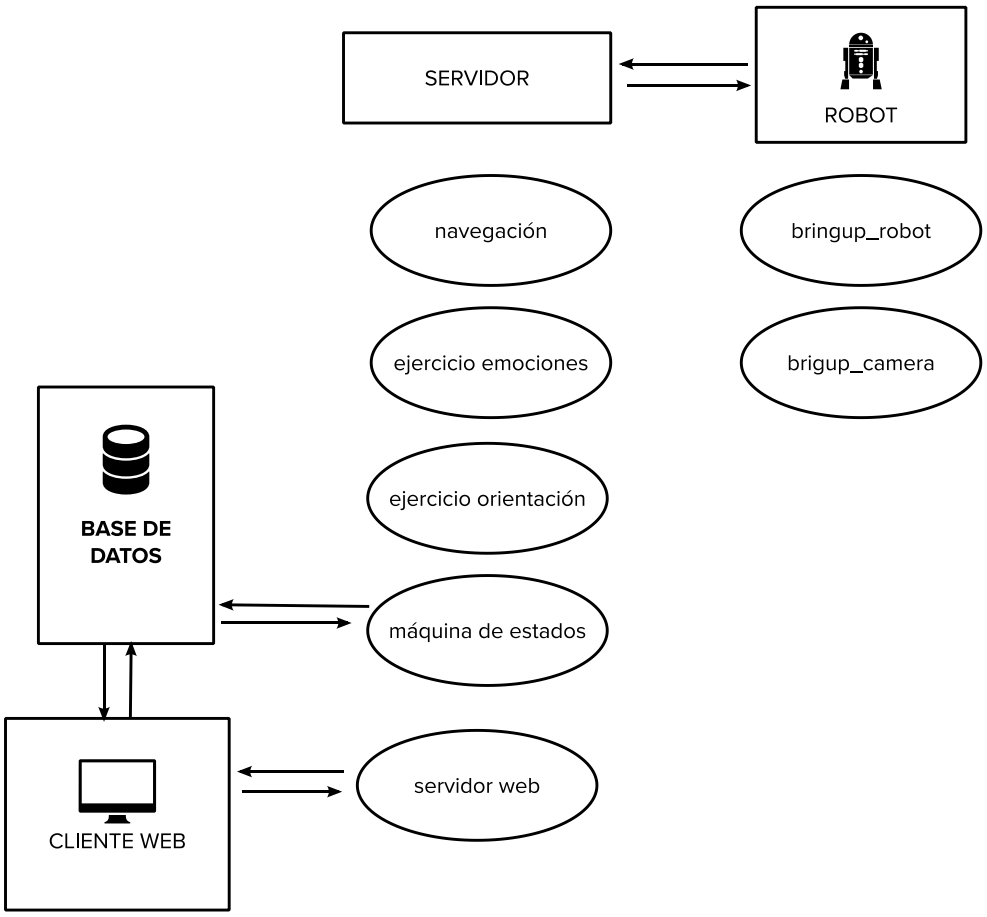


Figura 4: Esquema de comunicaciones del proyecto Jinkobot

Con el fin de delimitar las especificaciones se han redactado cuatro historias de usuario, dos con el psicólogo como actor y dos con el paciente.

Historias de usuario del psicólogo

Yo, como psicólogo, quiero poder iniciar un ejercicio desde la pagina web
Criterio de aceptación: • La aplicación web debe presentar un botón para iniciar el ejercicio • Al presionar el botón se tiene que llamar a la función correspondiente para iniciar los ejercicios • Se debe comprobar si el robot está conectado, si no lo está, antes de realizar la llamada se establecerá la conexión
Yo, como psicólogo, quiero poder ver la información de cada ejercicio en la pagina web
Criterio de aceptación: • Al pulsar sobre un paciente quiero ver una serie de gráficas • Las gráficas deben contener la última información disponible obtenida por el robot

Tabla 3: Historias de usuario del psicólogo

Historias de usuario del niño

Yo, como paciente, quiero que el robot me salude al iniciar el ejercicio
Criterio de aceptación: • El robot tiene que recibir el nombre del paciente. • El robot generar el audio y reproducirlo. • Se debe comprobar si el audio ya existe, y si así es, no lo volverá a generar solo lo reproducirá.
Yo, como paciente, quiero que al levantar una tarjeta con un pictograma me diga si mi respuesta es correcta o no
Criterio de aceptación: • Cuando se llegue al estado *Emotion* se debe realizar una llamada al servicio del ejercicio de emoción. • El servidor debe de estar en marcha • Tras un tiempo determinado si no se ha detectado nada, se dará la respuesta incorrecta y se pasará a la siguiente si es que la hay.

Tabla 4: Historias de usuario del niño

Para poder entender mejor estos requisitos se ha decidido dividirlos en tres partes, por un lado, tenemos toda la programación del robot, mediante ROS y Python, por otra el desarrollo de la aplicación web desde la cual el profesional podrá controlar el robot y visualizar toda la información relevante a las sesiones y ejercicios realizados por el niño. Y, por último, la parte del servidor que se encargará de almacenar todos los datos relativos al progreso del paciente.

Diseño de los paquetes de ROS

Tras las reuniones con el psicólogo se llegó a la conclusión de que los puntos más importantes en cuanto a ejercicios y rutinas eran la identificación de emociones y la orientación. Así que en eso se basarán los ejercicios disponibles.

Todo el proceso se realizará a través de una máquina de estados que guiará al niño a través de toda la interacción. El esquema de dicha máquina de estados es el siguiente:

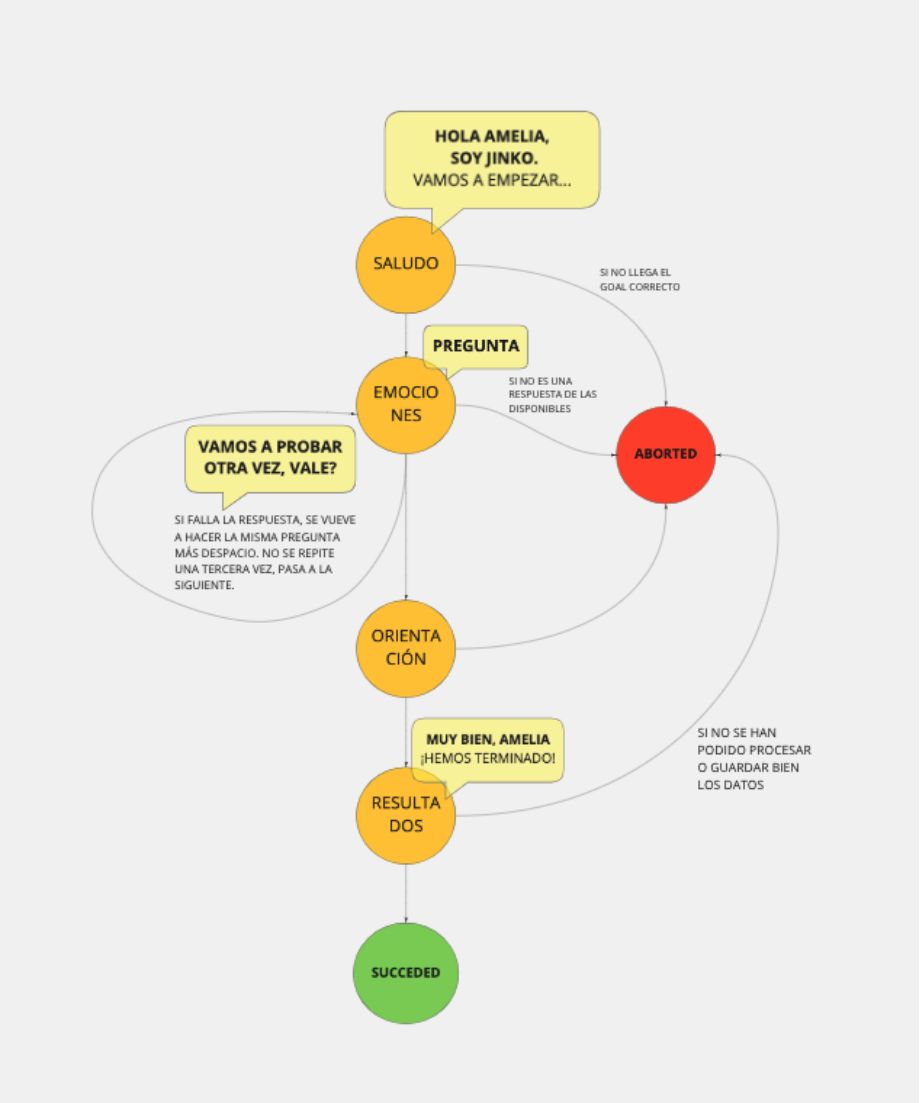


Figura 5: Diagrama de la máquina de estados Jinkobot

La máquina de estados se iniciará desde la aplicación web mediante una acción de ROS que incluirá el nombre del paciente, así como el índice de las preguntas a realizar y las coordenadas a dónde deberá dirigirse el robot en el ejercicio de orientación.

En el estado inicial el robot saludará al niño y pasará a realizar la primera pregunta, después de esperar el tiempo estimado para darle tiempo al niño a que levante la tarjeta, responderá si el niño ha acertado o no, si no ha acertado, el robot volverá a realizar la misma pregunta, pero más despacio. Esta acción sólo la realizará una vez, es decir, si la segunda vez, vuelve a fallar, se pasará a la siguiente pregunta. A continuación, se detalla el diagrama de flujo del ejercicio emociones.

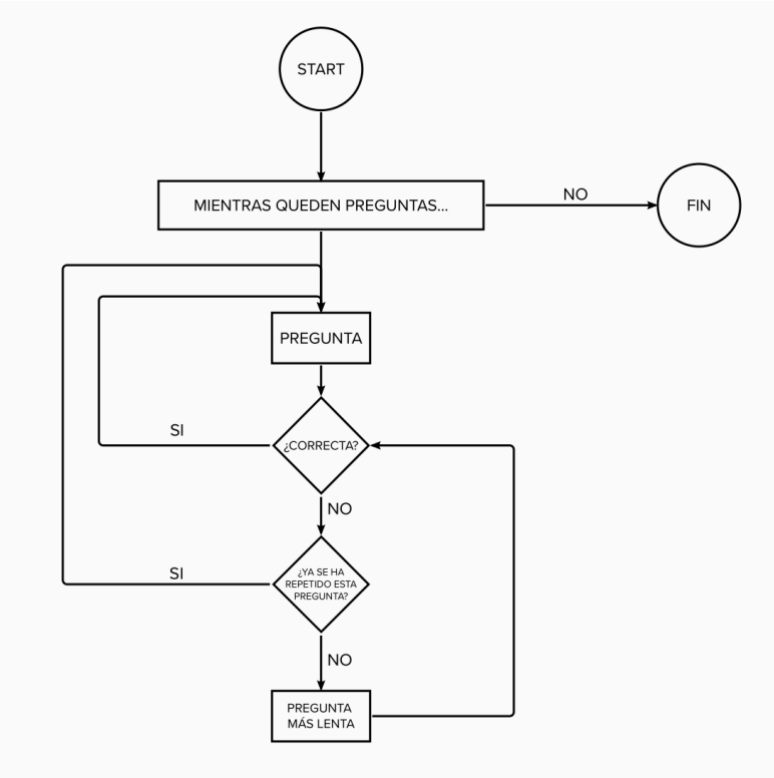


Figura 6: Diagrama de flujo del ejericio de emociones

Una vez terminadas las preguntas, se pasará al ejercicio de orientación, dónde el niño deberá seguir al robot a través de una trayectoria dada.

Para este último ejercicio se han establecido dos posibilidades, una ruta sencilla, en línea recta y otra con cambios de dirección en zigzag a través del espacio disponible.
El robot le pedirá al niño que le siga, y en el caso de la ruta en zigzag le irá animando a que continue siguiéndole antes de cada nuevo cambio de dirección.

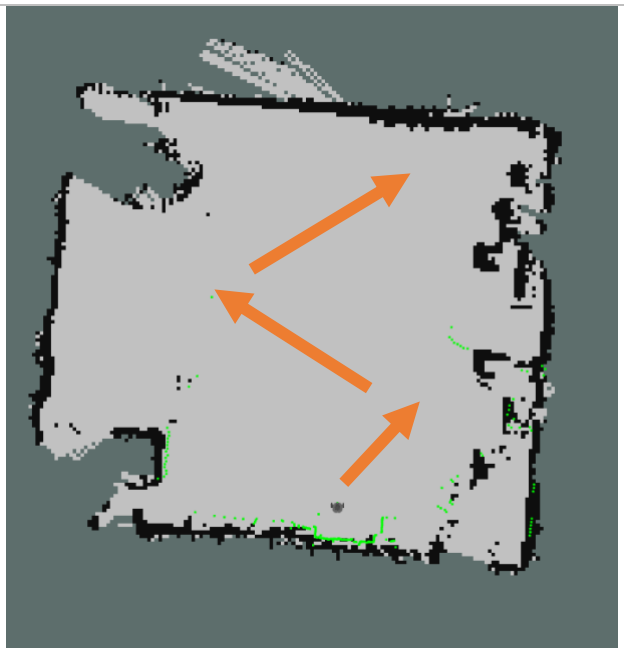
FÁCIL	INTERMENDIO
	

Tabla 5: Movimientos del ejercicio Orientación

Diseño de la aplicación web

Con el problema de la parte de servidor resuelta, vamos a entrar a detallar la aplicación web, se trata de una aplicación tipo *dashboard* dónde el profesional podrá iniciar sesión y ver y gestionar los pacientes asociados a su cuenta como los ejercicios llevados a cabo.

Wireframes y secciones

Login
El inicio de sesión de la aplicación se ha llevado a cabo mediante *Firebase auth*, servicio en la nube que proporciona seguridad y guardado de datos de inicio de sesión, así como recuperación de contraseña.

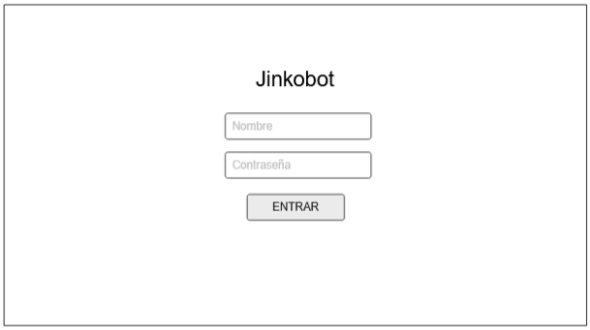


Figura 7: Wireframe: Página de inicio de sesión

Pacientes

La sección de pacientes muestra el listado de pacientes vinculados al usuario que ha iniciado sesión, desde esta misma sección se puede tanto entrar al detalle de cada paciente como crear un nuevo paciente introduciendo los datos oportunos.

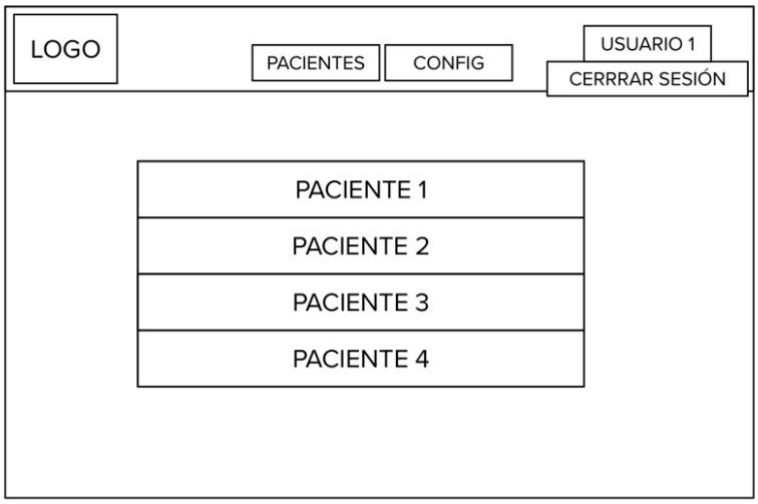


Figura 8: Wireframe: Página de pacientes

Detalle paciente

En la sección del detalle del paciente se puede ver la evolución del paciente, con los datos de sus sesiones, como son el tiempo empleado en ellas, la velocidad de respuesta, así como el porcentaje de acierto y fallo. Desde esta misma página se podrá acceder al inicio de los ejercicios, mediante un formulario el psicólogo podrá establecer tanto el número de preguntas como la dificultad y enviar la orden al robot para que empiece la sesión.

Una vez terminado este ejercicio, desde esta misma página el psicólogo o terapeuta se podrán añadir comentarios y/o observaciones sobre la sesión.

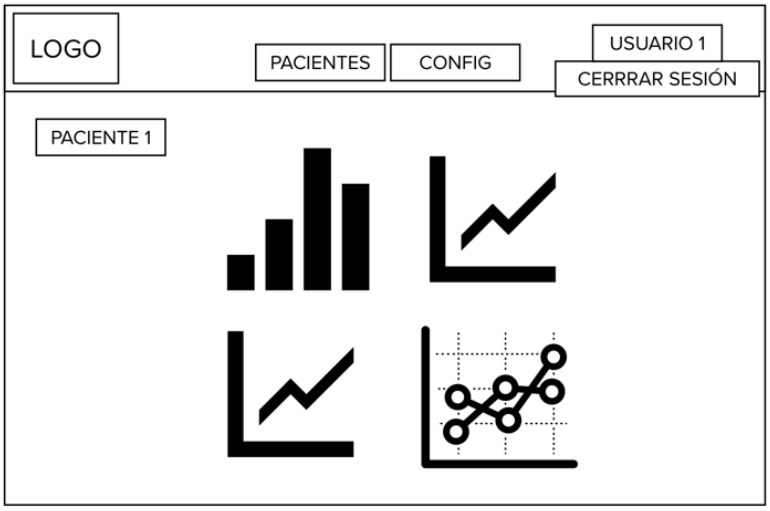


Figura 9: Wireframe: Página detalle de paciente

Desde esta misma pantalla al pulsar sobre el botón Empezar se abre una ventana emergente donde se pueden establecer los parámetros para realizar la llamada al robot y así empezar los ejercicios.

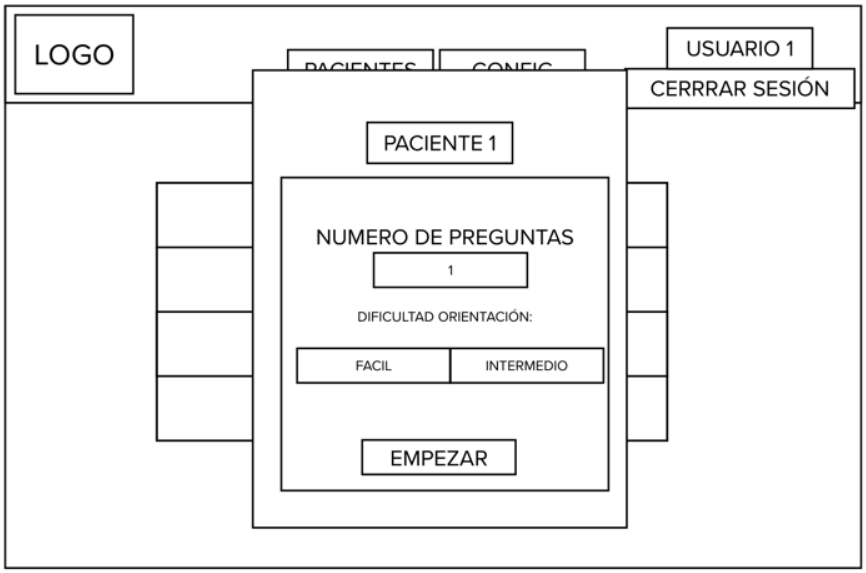


Figura 10: Wireframe: Configuración de los ejercicios

Configuración

Por último, contamos con la parte de configuración del robot, desde aquí se podrá hacer una prueba al robot para ver que todo funciona correctamente, también tendrá visión en tiempo real de la cámara del robot. Y también podrá empezarlos ejercicios.

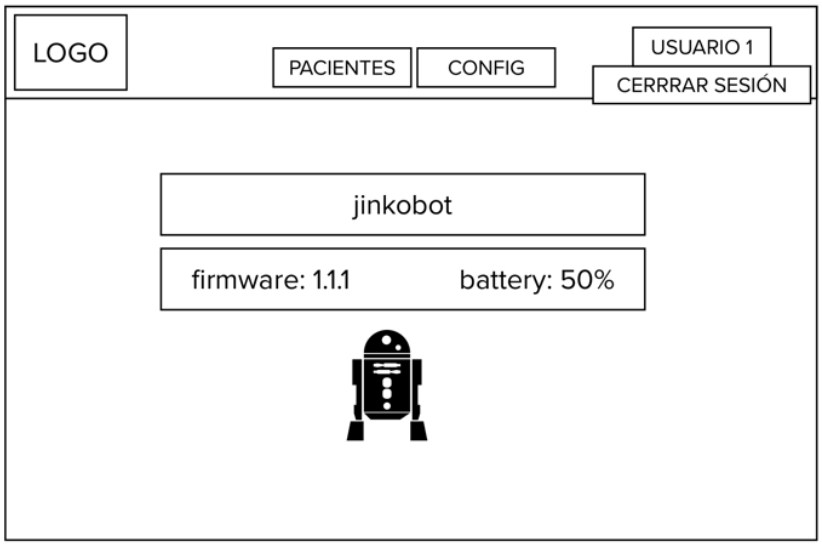


Figura 11: Wireframe: Página de configuración del robot

Diseño del servidor

Por motivos de envergadura de lo hasta ahora descrito se decidió por utilizar una arquitectura *serveless*, o lo que es lo mismo, hacer uso de alguna de las plataformas que ofrecen servicios propios de un servidor en la nube, tales como *Firebase, AWS, Azure…*

Por simplicidad y por tener una mayor experiencia durante el transcurso del grado, se decidió por Firebase, el servicio de Google en la nube que nos permite de una forma rápida cumplir con los siguientes requisitos de la parte del servidor:

- Autentificación mediante email y contraseña.
- Base de datos. En la figura 12 se puede observar el esquema de los datos de la base de datos.

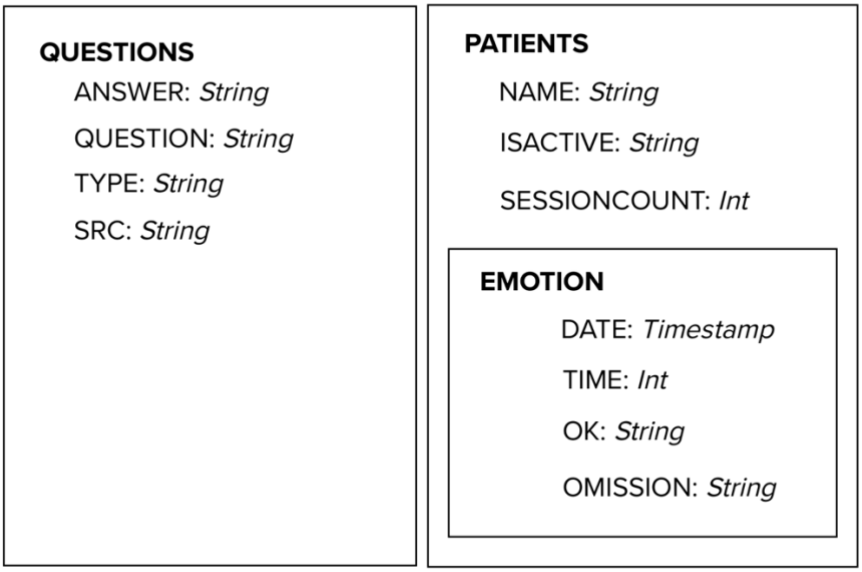


Figura 12: Diagrama de entidades para la base de datos

Capítulo 5

IMPLEMENTACIÓN

Vamos a pasar a detallar la implementación de cada uno de los paquetes del sistema de nodos en ROS que responde al siguiente esquema:

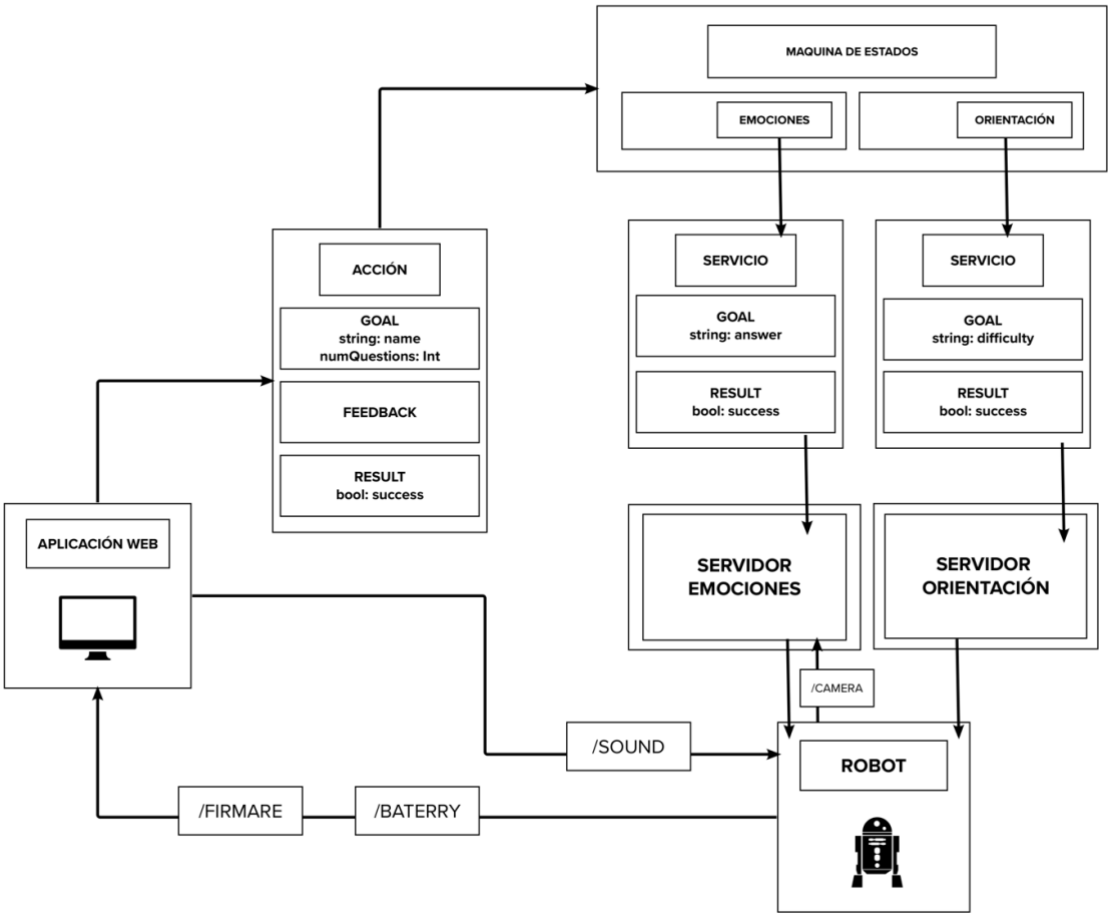


Figura 13: Esquema de comunicaciones en ROS

En el Anexo 4 se indican las direcciones de enlace a los repositorios remotos dónde se encuentra todo el código del proyecto.

También detallaremos en esta sección todo lo relacionado con la detección de imágenes mediante la librearía de Python *OpenCV* así como la implementación del servicio de Amazon Web Services *Polly* para la trasformación de texto a voz.

Contamos con tres paquetes de ROS, *jinkoaction*, el principal, y punto de inicio de todo nuestro sistema, y dos más, para los dos tipos de ejercicios descritos en la parte de diseño, emociones y orientación.

Paquete 1: JINKOACTION

El paquete *JinkoAction* es el punto de inicio de nuestra aplicación, se trata de una máquina estados que está a la espera de recibir información desde la aplicación web para empezar con la rutina de ejercicios.
Contiene también la configuración para los servicios de *Firebase* y *Amazon Polly*, y también servicios de comunicación con los otros dos paquetes, *Emotion* y *Orientation*.

Al tratarse del paquete con mayor envergadura del proyecto vamos a pasar a detallar su estructura detalladamente. Dentro del paquete principal de la aplicación podemos encontrar la estructura descrita en la figura 12.

Existen dos ficheros importantes en todo paquete de ROS. Éstos son necesarios para la correcta compilación del paquete. Contienen instrucciones que permiten cargar otros paquetes, librerías etc. Y demás dependencias necesarias. Estos ficheros deben estar ubicados en la carpeta raíz del proyecto. Pasamos a detallarlos:

- package.xml

  Este archivo contiene los metadatos del proyecto, el nombre, la versión, los autores etc. Además de las dependencias de otros paquetes.

- CMakeList.txt

  En este archivo se especifica toda la información necesaria para la compilación del proyecto. Dependencias de otros paquetes, librerías necesarias para su correcta compilación.

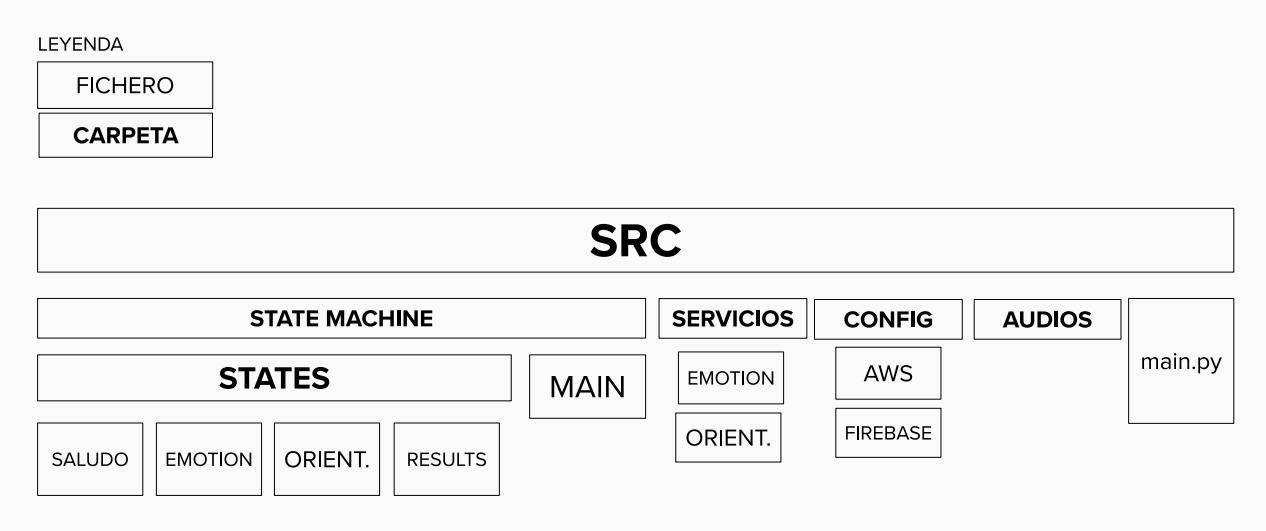


Figura 14: Estructura del proyecto de ROS

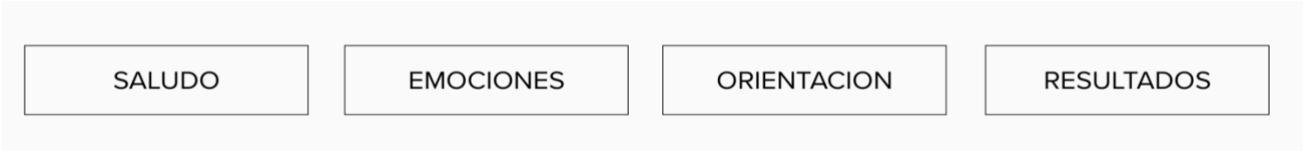
En la carpeta *src* se encuentra todo el grueso de la implementación, a su misma altura podemos ver la carpeta *action* que tiene el mensaje *jinko.action* utilizado para lanzar la acción que controla la ejecución en el robot y que se explicará más adelante.
Dentro de la carpeta *src*, podemos encontrar los siguiente:

- La carpeta *audios* contiene todos los audios generados por el servicio *Amazon Web Service Polly*.

- En la carpeta *config* están almacenadas las claves de acceso y las configuraciones tanto del AWS como al servicio *Firebase* de Google, servicio que nos permite tener una base de datos en la nube.
  Estas credenciales serán utilizadas en la siguiente carpeta para poder acceder a dichos servicios.
  Cabe destacar también que estos ficheros están omitidos dentro del archivo .gitignore para no compartir esta información al hacer uso del servicio de repositorios remoto github.

- En la carpeta *services*, como acabamos de mencionar, nos encontramos con dos clases que nos servirán de puente para poder acceder a los servicios tanto de AWS *Polly*, para pasar texto a habla natural, como de *Firebase*, para poder almacenar datos en nuestra base de datos, así como leer de la misma para tener toda la información relativa a las preguntas que el robot debe realizar al paciente.

- En la carpeta *StateMachine* se encuentra ellos ficheros principales de toda la implementación. Partimos de un script que contiene la clase StateMachineClass.py que crea una máquina de estados con las 4 fases de nuestro ejercicio.



Dentro de esta carpeta encontramos también una subcarpeta llamada *states* dónde encontraremos las clases correspondientes a cada uno de los estados, a saber, y en orden de ejecución, Grettings.py, Emotion.py, Orientation.py y Results.py. En la figura 13 se puede ver el diagrama de clases correspondiente.

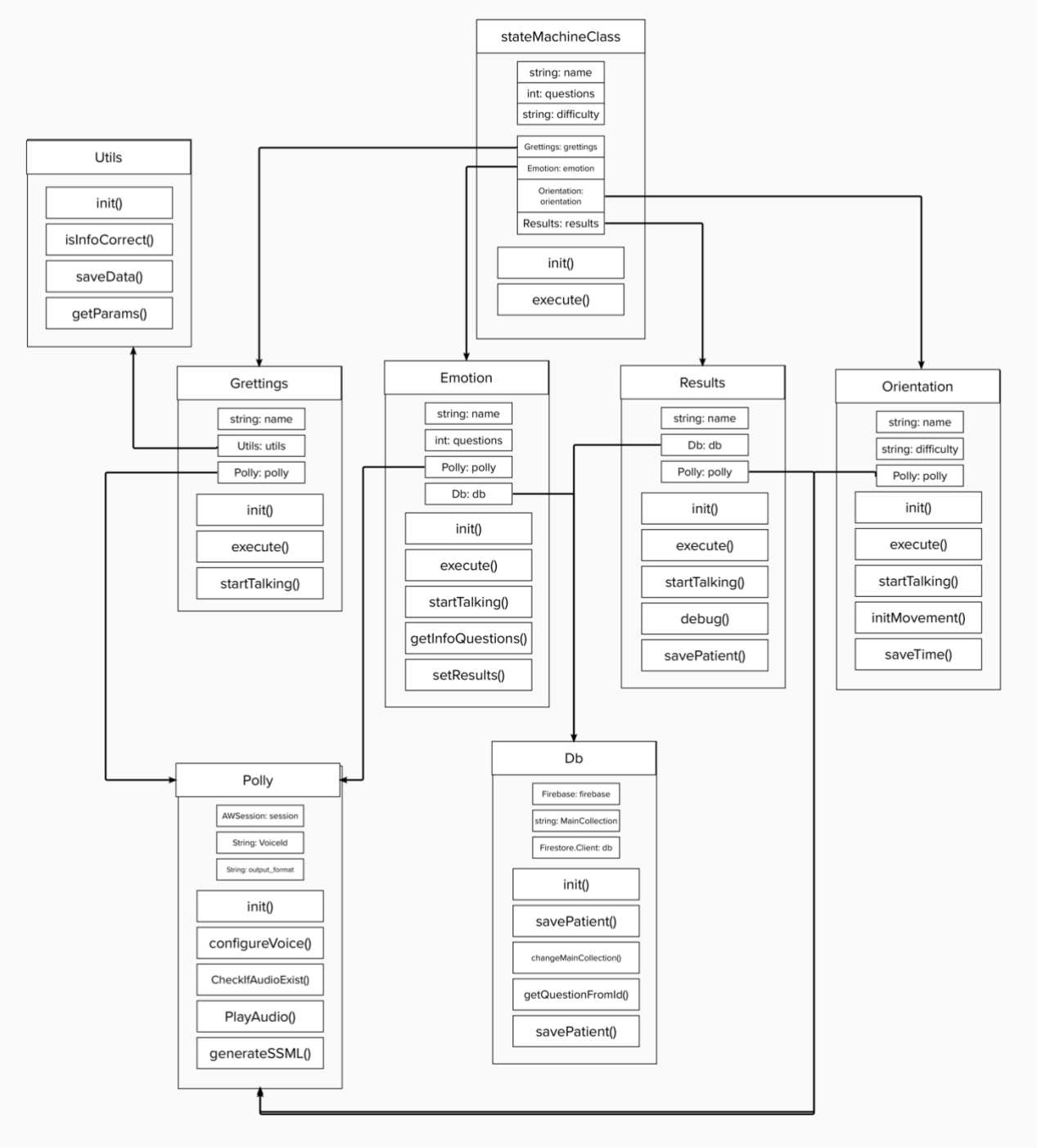


Figura 15: Diagrama de clases del paquete JinkoAction

- En la carpeta ***rosclients*** tenemos dos archivos, que serán los encargados de conectarse con los paquetes que ejecutan cada uno de los ejercicios.

Y ya por último nos encontramos con el punto de inicio de la implementación, se trata del archivo main.py que es el punto de inicio a nuestra máquina de estados, eje principal de la implementación en ROS.

Dentro de nuestro fichero *main*, lo que hacemos es valernos de la librería *actionlib* y de su función *SimpleActionServer* para crear un servidor que se mantiene a la escucha esperando una llamada que inicie la ejecución del programa.

Este método recibe como primer parámetro el nombre identificativo del servidor, es decir, el nombre que tendrá que saber el cliente para poder establecer la conexión.

Como segundo parámetro recibe el mensaje, es decir la estructura del tipo de información que van a compartir.

Como tercer parámetro este método necesita la función que se va a ejecutar cada vez que el servidor reciba una llamada. Aquí es donde empieza realmente la ejecución del programa. Esta función de *callback* permite recibir como parámetro de entrada la información que le hemos enviado desde la aplicación web. Esta información es la siguiente:

- El nombre del paciente para conseguir los audios personalizados, pero también para poder guardar después los datos obtenidos en la base de datos
- El número de preguntas del ejercicio de emociones
- La dificultad del ejercicio de orientación

Como último parámetro recibe un booleano que representa la posibilidad de iniciar el servidor de manera automática sin necesidad que reciba una llamada. En este caso, al no ser lo que pretendemos, su valor será el de false.

El parámetro de salida o resultado de esta llamada viene ligado completamente a la máquina de estados. Cada estado tiene una salida denominada *aborted* que será lanzada en caso de que se encuentre algún error en la ejecución.

Por ejemplo: Si al iniciar la máquina de estados, en el estado *Grettings* el nombre del paciente no es un *string* válido, la clase *Grettings* devolverá *aborted* y automáticamente la acción se dará por terminada devolviendo el valor false anteriormente mencionado.

Lo que nos quedaría una vez establecido nuestro servidor, es iniciarlo mediante el método *server.start()* y por último hacer uso de la librería *rospy* para dejar al servidor ejecutándose y en espera de la llegada de alguna llamada. Esto se hace gracias al método *rospy.spin()*.

Paquete 2: EMOTION

El paquete emociones es el encargado de poner la cámara del robot en funcionamiento y mediante una aplicación de procesamiento de imágenes detectar la tarjeta que ha levantado el niño para saber si es la respuesta correcta a la pregunta.

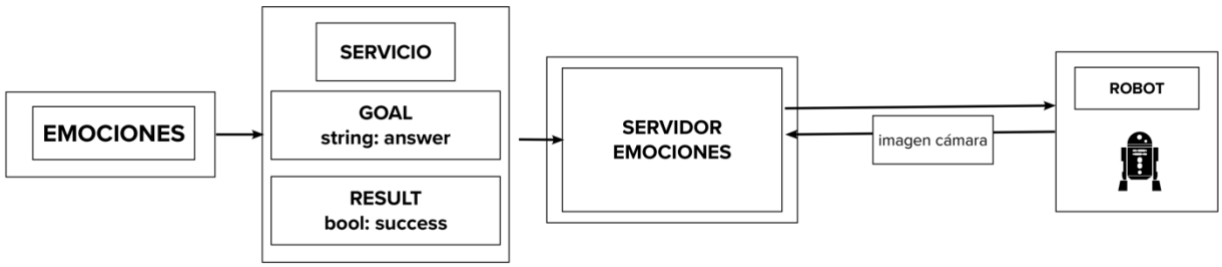


Figura 16: Esquema paquete emociones

El paquete está a la espera de recibir la respuesta correcta en forma de string, por ejemplo “triste”. Después de recibir este parámetro mediante visión artificial detectará la tarjeta que ha levantado el niño y compara este resultado con la respuesta correcta recibida para saber si el niño ha acertado.

Como se puede comprobar la parte más importante de este paquete es la detección de imágenes, para ello se ha hecho uso de una de las librerías de visión artificial más importantes, por usada y por potente, de Python, su nombre, *OpenCV*.

Reconocimiento de imágenes mediante OpenCV

En un primer momento se optó por la utilización de la función *templateMatching* que realiza una búsqueda de una imagen patrón en otra.

Pero este método ha presentado diferentes dificultades que ha hecho imposible su implementación óptima.

- A tratarse de una imagen sacada desde un video en tiempo real, la iluminación juega un papel muy importante a la hora de realizar el reconocimiento. Al acercar la tarjeta demasiado a la webcam, la propia luz de la pantalla daba como resultado coincidencias erróneas.
- Teniendo en cuenta que hay que mostrar la tarjeta hacia la cámara, era imposible evitar un ligero movimiento, así como alguna inclinación en la tarjeta que a veces daba como resultado una imagen un poco borrosa que ocasionaba, también, malos resultados.

Es por esta razón que se realizó una búsqueda de diferentes algoritmos para encontrar semejanzas entre imágenes, dando con uno idóneo para nuestro propósito.

A continuación, se detallan los pasos a seguir:

- Una vez pasadas las dos imágenes a B/N (webcam y plantilla) utilizamos el objeto ORB (*Oriented FAST and Rotated BRIEF*) para detectar los *keypoints* y descriptores a comparar de ambas imágenes.

```
detector = cv2.ORB_create(1000);
kp1, desc1 = detector.detectAndCompute(frame, None)
kp2, desc2 = detector.detectAndCompute(pattern, None)
```

Figura 17: Detección de keypoints con OpenCV

- FLANN (*fast approximate nearest-neighbor searches)* es una librería que utiliza un algoritmo de búsqueda a partir de los vecinos más cercanos encontrando coincidencias entre dos imágenes dadas.

```
# Creación del detector
matcher = cv2.FlannBasedMatcher(index_params,search_params)

# Búsqueda de coincidencias a partir de los descriptores hallados con ORB
matches = matcher.knnMatch(desc1, desc2, 2)
```

Figura 18: Utilización del alogoritmo de los vecimos más cercanos

- Los resultados se filtrarán en base a un umbral.

```
umbral = 0.7
good_matches = [m[0] for m in matches \
            if len(m) == 2 and m[0].distance < m[1].distance * umbral]
```

Figura 19: Aplicación de un umbral a los coincidencias encontradas

- Para tener en cuenta los cambios de perspectiva se guardan los puntos dónde ha habido coincidencias en dos vectores, uno para la imagen fuente y otra para la plantilla.

- A partir de esta información crea una nueva matriz y una máscara que contempla los cambios de perspectiva con la función *findHomography*.

```
mtrx, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
```

Figura 20: Reconocimiento de los cambios de perspectiva

- Después de esto comprueba que la suma de las coincidencias (teniendo en cuanta las transformaciones de perspectiva) sea mayor que el número mínimo de coincidencias que queramos tener (especificado por nosotros).

Paquete 3: ORIENTATION

El paquete orientación se encarga de recibir unas coordenadas y mover al robot hacia esas coordenadas mediante navegación autónoma.
Para la implementación de este ejercicio lo primero fue delimitar el área de acción del robot, es decir, la zona por la que el robot se moverá.

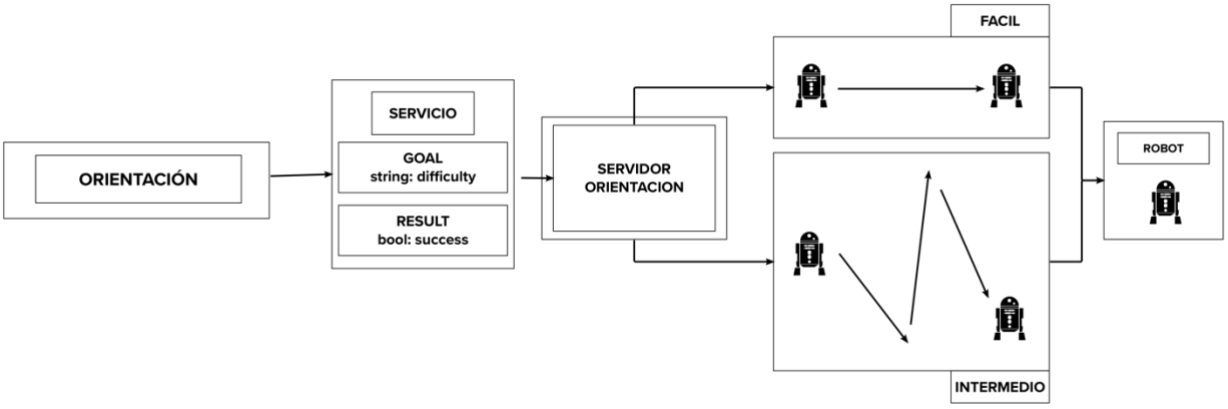


Figura 21: Esquema paquete orientación

Mediante el paquete SLAM [8] y su funcionalidad es posible mapear una zona concreta del espacio mediante el robot. Una vez mapeado se guarda esta información en forma de dos archivos. Estos dos archivos contienen toda la información relacionada con el mismo, como, por ejemplo, su eje de coordenadas del mismo. Estos archivos son muy importantes ya que serán necesarios para cargarlos en el sistema y poder enviarle al robot unas coordenadas de destino y que éste las entienda correctamente.

No se ha implementado una función de mapeo automático ya que se ha preferido dar ese servicio ya preparado al terapeuta, de manera que se le entrega el robot totalmente configurado.

Una vez tenemos la zona mapeada mediante el programa RVIZ [9], desde el cual podemos ver el mapa en el ordenador, podemos seleccionar unas coordenadas de destino para que robot empiece su movimiento.
Para simplificar esta funcionalidad se ha creado un paquete llamado *my_navegation_stack* que realiza todas las acciones necesarias para que el ejercicio de orientación funcione. Este paquete permite:

- Cargar el mapa que hemos mapeado en el sistema
- Lanzar los paquetes propios del robot necesarios para la navegación
- Y por último abrir el programa RVIZ para poder ver el mapa y al robot en él

Ahora ya podemos mandar al robot a unas coordenadas específicas. Dada la dificultad seleccionada el robot realizará los siguientes movimientos:

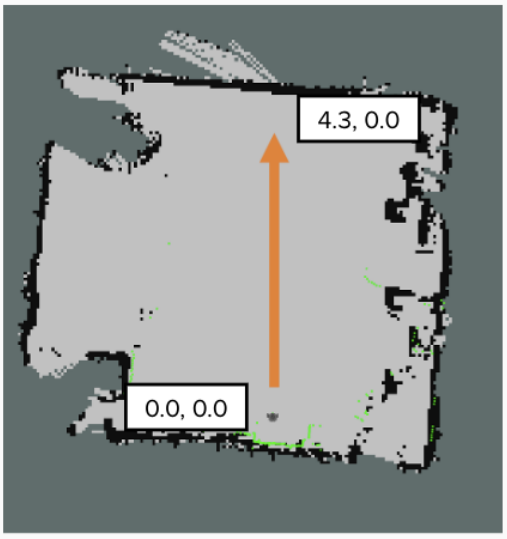
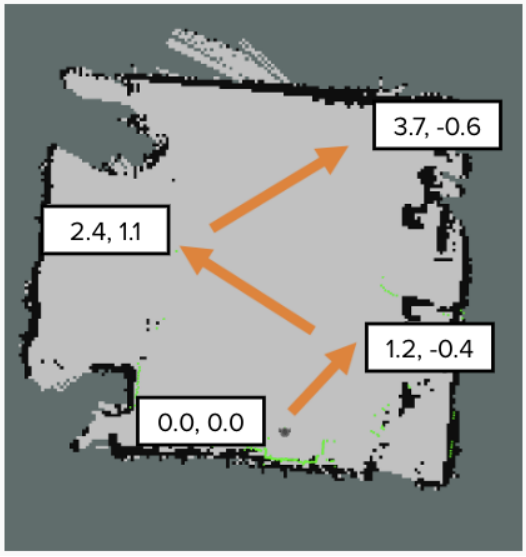
FÁCIL	INTERMENDIO
	

Tabla 6: Movimientos del ejercicio Orientación

MÉTODOS DE COMUNICACIÓN

A continuación, se pasan a detallar los sistemas de comunicación, no solo entre los paquetes anteriormente descritos, también con la aplicación web. Los sistemas de comunicación utilizados han sido dos, acciones y servicios, básicamente su función es la misma, pero tienen características diferentes que se detallan en la tabla 1.

	SERVICIO	ACCIÓN
Extensión del archivo del mensaje	*.srv*	*.action*
¿Se puede cancelar?	no	si
¿Tiene retroalimentación?	no	si
Múltiples parámetros de entrada	si	si
Múltiples parámetros de salida	si	si

Tabla 7: Comparativa entre acciones y servicios en ROS

ACCION

La acción implementada para este proyecto representa la comunicación principal entre la aplicación web y el paquete *JinkoAction*. Desde la aplicación web el terapeuta es el encargado de establecer la configuración para la ejecución de los ejercicios, a saber:

- El número de preguntas del ejercicio de emociones.
- La dificultad del ejercicio de orientación.

Pero esta no es toda la información que se compartirá con el paquete de ROS, a parte de la configuración anteriormente descrita, también se le pasará el nombre del paciente, para conseguir lo siguiente:

- Audios del robot personalizados.
- Guardar los datos obtenidos al final dentro del paciente correcto en la base de datos.

Para poder compartir la información anteriormente descrita se necesita un archivo de extensión *.action.* Una cosa importante a tener en cuenta con los archivos con extensión *.action*, y también con los .srv, es que deben estar registrados en el archivo CMakeLists.txt de la siguiente manera:

Figura 22: archivo .action añadido al archivo CMakeLists.txt

Para poder declarar nuestro mensaje en dicho archivo, éste tiene que estar dentro de nuestro paquete.

También podemos crear nuestro mensaje dentro de otro paquete, pero en este caso el mensaje tiene que estar declarado en el package.xml, tanto como dependencia de ejecución como de construcción, para después poder añadirlo como dependencia externa dentro de CMakeLists.txt.

SERVICIOS

La manera de comunicarnos con los paquetes que contienen la ejecución de los ejercicios es mediante servicios.

De la misma manera que con las acciones, es necesario, igualmente, declarar el archivo .srv en el CMakeLists.txt, dependiendo de si están en el propio paquete o en uno externo.

Necesitamos importar el mensaje *.srv* y crear una función que será llamada desde el estado correspondiente de la máquina de estados.
El cliente, a través del paquete *rospy.ServiceProxy*, realizará la llamada pasando como parámetro lo siguiente:

- El nombre del paquete a ejecutar, este paquete de ROS estará activo y a la espera de recibir la llamada.
- El tipo de mensaje de comunicación que compartirán, esta es la información contenida en el archivo .srv.

SERVICIO EMOTION

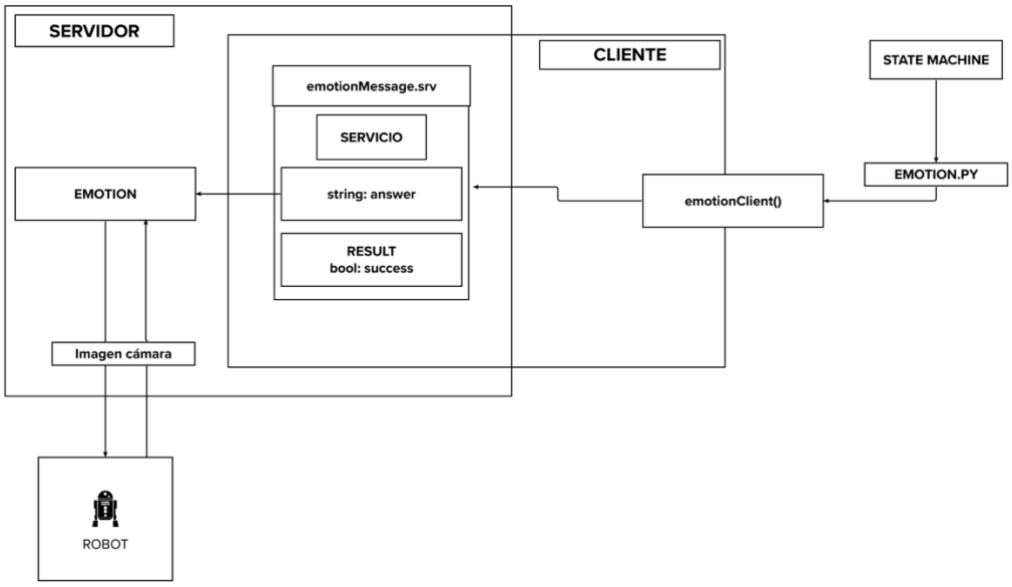


Figura 23: Esquema de comunicación del estado Emotion con el robot

En este ejemplo al tratarse del cliente para el paquete de emociones, será llamado desde el segundo estado de nuestra máquina de estado Emotion.py de la siguiente manera:

```
output = emotionClient(self.answer)
```

Como se puede observar esta función recibe un parámetro llamado *answer* que contiene la respuesta correcta al ejercicio, y es con la que comparará la respuesta del niño para saber si ha acertado o no.

SERVICIO ORIENTATION

Para el caso del servicio de orientación se ha seguido el mismo procedimiento, como se puede observar el la figura 16, contamos con un cliente que al ejecutarse llama al servicio de orientación para poner en marcha al robot y establecer las coordenadas a las que debe desplazarse. La información compartida es la dificultad del ejercicio. Una vez recibida la dificultad se establecen las coordenadas correspondientes y el robot empieza su movimiento.

En el caso de la dificultad fácil, solo realiza una llamada al servicio de movimiento, y al terminar recibe un parámetro booleano para saber si las coordenadas finales concuerdan con las coordenadas establecidas como destino.

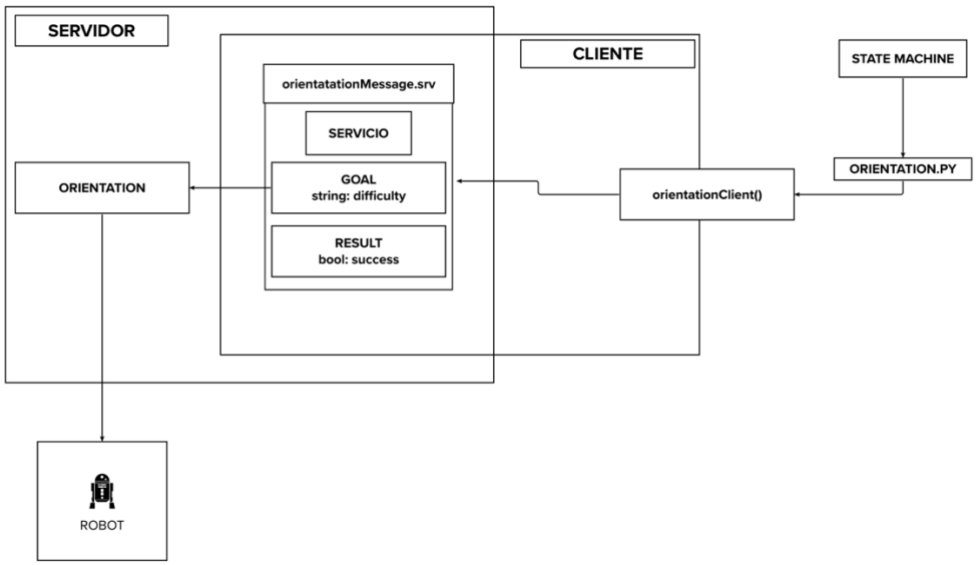


Figura 24: Esquema de comunicación del estado Orientation con el robot

En el caso de la dificultad intermedia, este servicio es llamado tres veces, para conseguir, de esta manera, un movimiento en zigzag añadiendo la posibilidad al robot de parar entre destinos y poder animar al niño mediante audios a que le siga.

Amazon Polly

Y por último dentro de la implementación del robot cabe destacar la utilización de *Amazon Polly* para conseguir que el robot se comunique con el niño mediante voz.

Para ello se ha creado una clase en Python que se conecta con el servicio de Amazon en la nube.

Esta clase recibe el texto que se quiere convertir a voz y comprueba si el fichero de voz ya ha sido creado con anterioridad.

Por ejemplo, si es la segunda sesión con el niño, el audio relacionado con el saludo ya se habría generado la primera vez y solo sería necesario reproducirlo.

Esta comprobación también es importante ya que el servicio presenta una capa gratuita limitada, que a partir de cierto número de caracteres procesados pasa a ser de pago.

Si no existe dicho fichero el texto es enviado al servicio de Amazon y este genera un archivo de audio .mp3 que se será almacenado en la carpeta audios del proyecto, y tras esto lo reproducirá.

Para la utilización de este servicio es necesario contar con una cuenta de desarrollador en *Amazon web services* y dar de alta nuestro proyecto en *Amazon Polly* para conseguir las credenciales necesarias para poder llamar al servicio desde nuestra aplicación en Python.

Implementación de la aplicación web

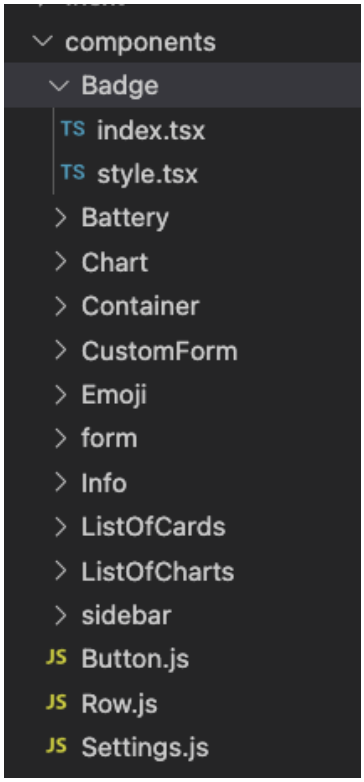


Figura 25: Listado de componentes de la aplicación web

En la figura 17 podemos ver una lista de los componentes utilizados en la nuestra aplicación web:
Una de las grandes facilidades que nos ofrece React es la de poder reutilizar componentes, lo que evita en gran medida la duplicación de código.
Por ejemplo, se puede usar el componente *Button.js* en cualquier parte de la aplicación solamente importándolo y definiéndolo de la siguiente manera:

```
<Button handle=”someFunction” bg=”orange” />
```

De esta manera gracias al a propiedad *handle* podremos pasarle la función a ejecutar cuando se pulse el botón y decidir también el color de fondo de este.

Para la correcta conexión de la aplicación ROS y el grosso del proyecto se han desarrollado dos servicios dentro de la propia aplicación.

- Servicio para la conexión con el servidor en la nube de *Firebase*
- Servicio para la comunicación con ROS mediante la librería roslib.js

Antes de pasar a desarrollar estos dos servicios vamos a explicar brevemente las funcionalidades que nos ofrecen Firebase y la librería roslib.js.

Firebase

Gracias a *Firebase* podemos autentificar al usuario y evitar que se pueda acceder a la información alojada en la base de datos. Los pasos que seguir son los siguientes:

- Obtener las credenciales necesarias desde la consola de administración de *Firebase* para poder acceder al servicio.
- Importar dichas credenciales en nuestro proyecto.

- Y, por último, importar los servicios que vamos a utilizar, en este caso la base de datos y la autentificación.

Cabe destacar que el archivo con las credenciales debería estar incluido en el archivo .gitignore para, así, evitar que dichas claves privadas para nuestro proyecto sean incluidas en nuestro repositorio de *github*.

Librería roslib.js

El uso de la librería roslib.js nos permite ejecutar las siguientes funcionalidades:

- Lanzar la llamada a la acción que iniciará todo el proceso de ejercicios
- Suscribirse a la información del robot, a saber, el nivel de batería, la versión de firmware, y también la información relativa a si el robot está conectado a la corriente o no. Esto se realiza desde la página de configuración, donde también se podrá conectar al robot en el caso de que no lo estuviera.

Para conseguir esto se ha creado un fichero que presenta las funciones correspondientes para realizar estas acciones y así poder ser importado desde cualquier parte de la aplicación dónde sea necesario.

Implementación del servidor

Como se ha comentado en la sección de diseño el servicio elegido para implementar la parte del servidor de este proyecto ha sido *firebase*.
Desde su panel de control se podía comprobar la correcta inserción y modificación de datos tanto desde el robot como desde la aplicación web.
También se podían insertar los primeros datos manualmente para comprobar la correcta conexión y listado de datos en la aplicación web.

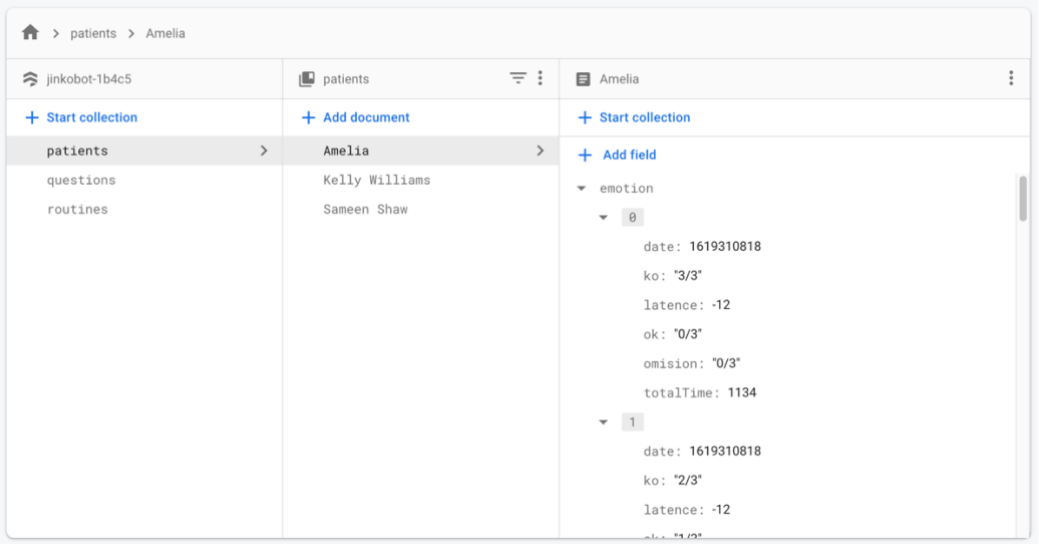


Figura 26: Captura del panel de gestión de la base de datos

Autenticación

Una de las ventajas que ofrece servicios como el de *Firebase*, es la implementación de un *login* de manera sencilla. Como se puede ver en la siguiente captura de la plataforma, *Firebase* cuenta con una integración sencilla con la mayoría de las aplicaciones más usadas en estos momentos.

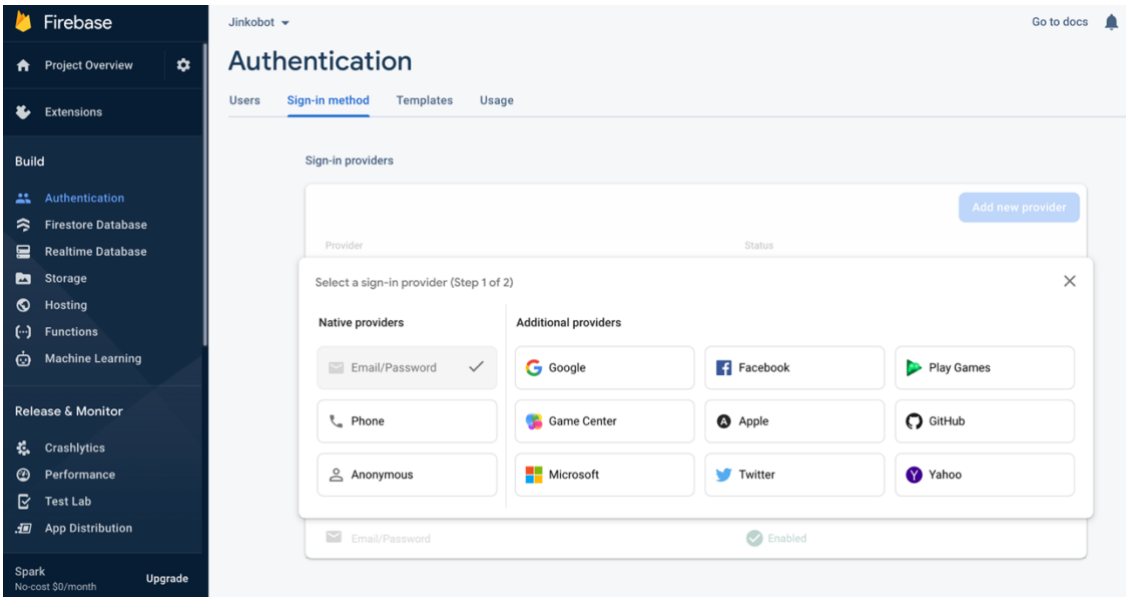


Figura 27: Panel de gestión de autentificación en Firebase

Para este proyecto, se ha decidido por el sistema de autentificación mediante email y contraseña como se puede comprobar en la captura superior al estar marcada esa primera opción.
En el apartado anterior ya se ha explicado la integración de este sistema en la aplicación web, pero también hacemos uso de la base de datos desde la aplicación de ROS.

Desde el servidor ROS la conexión con la base de datos se utiliza para dos tareas fundamentales.

- Obtener el enunciado y la respuesta correcta de las preguntas a realizar
- Guardar los resultados obtenidos en el último estado de la máquina de estados

Para ello se ha creado un servicio, es decir, un archivo que permite mantener una sesión abierta con la base de datos en el servicio de Google en la nube. A este servicio se accede desde el estado EMOTION y desde RESULTS, como ya hemos explicado anteriormente.

La forma de incluir el acceso a *Firebase* desde Python es similar a la explicada anteriormente desde la aplicación web, obteniendo las credenciales y demás pasos, pero esta vez accediendo solo a la base de datos ya que el servicio de autentificación solo es necesario en la aplicación web.

Del mismo modo, hay que incluir el fichero de credenciales en el *.gitignore.*

Capítulo 7

TESTEO

En este apartado vamos a presentar de una de las partes más importantes de una aplicación, el testeo. Tanto de la aplicación web como de la aplicación en ROS.

Antes de que un producto o servicio tecnológico salga a producción y esté en contacto directo con los usuarios finales debería haber pasado una exhaustivo set de test que garanticen en la mayor medida posible su correcto funcionamiento.
Aunque conseguir un correcto funcionamiento al 100% es una utopía, cuantas más parte de tu aplicación tengamos aseguradas mediante test más errores y futuros problemas conseguiremos evitar.

Una metodología muy interesantes sobre esto es TDD (Test Driven Development). Esta forma de trabajar consisten en realizar primero el test de la funcionalidad que se pretende implementar y ejecutarlo para ver como falla, porque todavía no hemos programado nada, una vez hecho esto, se implementa la funcionalidad y se vuelve a ejecutar el test hasta que este da como resultado que todo funciona correctamente.

Esta forma de trabajar requiere de una mayor inversión de tiempo al a hora de programar, y esto le ha valido una buena serie de detractores dentro del mundo del desarrollo.
Pero entre sus beneficios también se encuentra que mediante estos test, que se han escrito antes que nada y nos han hecho pensar que es lo que queremos conseguir antes de empezar a escribir código, también representa una documentación adicional al proyecto.

Testeo de la aplicación web

Para testear la aplicación web se ha decido utilizar Jest [10] junto React Testing Library [11].
Jest es un test runner o también llamado test framework que permite gestionar la ejecución de los test y react testing library es una colección de utilidades especificas para React.
Se trata de dos librerías complementarias que facilitan en enorme medida el testeo de cualquier aplicación web realizada con estas herramientas.
Vamos a explicar el testeo de la página principal, donde nos encontramos un texto de bienvenida y un formulario para iniciar sesión como se ve a continuación:

Un buen comienzo para nuestra batería de pruebas es comprobar que el contenido se ha renderizado correctamente, para ello vamos a comprobar si el texto de bienvenida se ha renderizado correctamente y se puede leer en nuestra página de inicio.

```
import Home from "../pages/index";
import "@testing-library/jest-dom";
import { render } from "@testing-library/react";

test("render content", () => {
    const { getByText } = render(<Home />);
    const title = getByText("Welcome to Jinkobot");
    expect(title).toBeInTheDocument();
})
```

Figura 28: Código para el testeo de la página de inicio

De lo que se trata es de renderizar el componente Home, que es donde se encuentra nuestro texto de bienvenida, y comprobar si se puede encontrar el texto de bienvenida en ese componente recién renderizado.
Una aproximación al testeo es pensar como si se tratase de un usuario, y preguntarse ¿El usuario puede ver la página de inicio correctamente? Y de esta manera sabemos que, al menos, el texto de bienvenida aparece sin problemas en nuestra aplicación.

Pero como también hemos comentado también tenemos un formulario de inicio de sesión, y esta parte sí que es crítica para el correcto funcionamiento de nuestra aplicación, sin ella no se podría acceder a todas las funcionalidades implementadas.
Así que, de nuevo nos ponemos en la piel del usuario y vamos a ver que debería pasar cuando éste introduce su email y su contraseña y presiona el botón de *login*.
Lo sabemos, la aplicación debería redirigirnos a la página de inicio del usuario que acaba de iniciar sesión.
Siguiente la metodología TDD vamos a crear un test que simule este comportamiento, añadimos un email y una *password* y clicamos en el botón de *login*.
Este el resultado obtenido, el test ha fallado porque no ha encontrado el texto “Charly Brown” en ningún parte de la página.

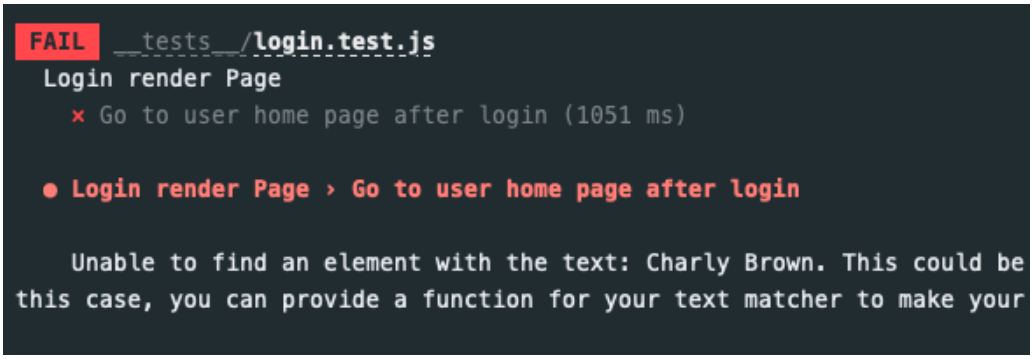


Figura 29: Captura del test fallando

Una buena práctica a la hora de testar una aplicación, y que con TDD viene ya intrínsecamente, es la de ver como nuestras pruebas fallan, de otra manera, la prueba puede estar dando un resultado en positivo por diferentes razones a las que podríamos pensar e un primer momento.

Ahora solo nos faltaría implementar la lógica que permita al usuario iniciar sesión y llegar a su página de inicio y este pasará de rojo a verde.

Así de esta manera una vez tengamos todas nuestras funcionalidades implementadas podremos ver todas nuestras pruebas en verde.

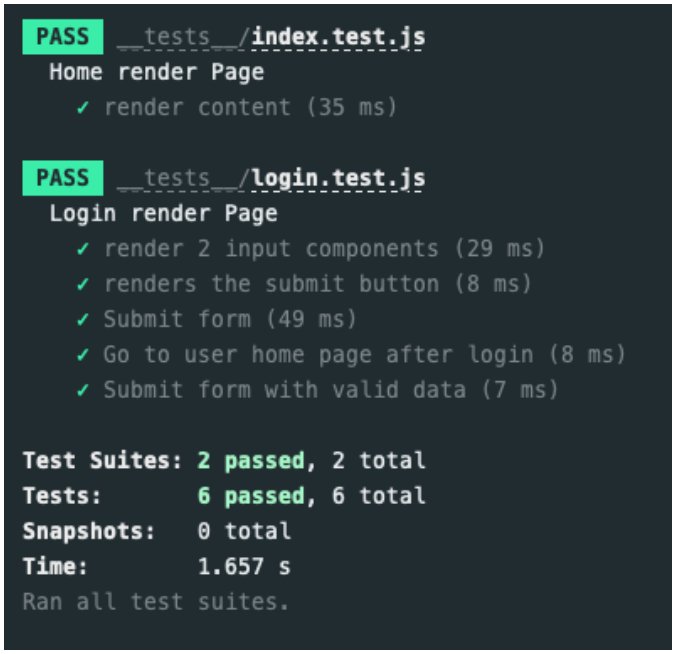


Figura 30: Captura de los test de la página home pasando

Testeo de la aplicación ROS

Uno de los problemas ante los que se enfrenta un sistema como ROS es la falta de información y tutoriales presentes en internet, si lo comparamos con otras tecnologías como el desarrollo web.

Por suerte cuenta con Python como uno de sus lenguajes de programación y esto ayuda bastante, ya que Python es un lenguaje que está de moda por todo lo relacionado con la ciencia de datos y cuenta con mucha información y publicaciones muy interesantes sobre cómo desarrollar buenos test.

Unnitest es un framework de testing para Python, y es el elegido para testear nuestra aplicación.
Los tests basados en esta librería son compatibles con ROS siempre se añado un wrapper que ayude a crear el output correcto. Existen dos tipos de test soportados.

Los llamdos *ROS-Nodel-Level Integration Tests*. Se encargan de testear topics o servicios. En este caso el script del test, es decir el archivo que testea el topic o el servicio es un nodo de la aplicación. Y *los Code-Level Unit Tests* que realizan llamadas directas al código. El script de test no es un nodo. Hay que distinguir bien entre los dos tipos, ya que la sintaxis de ejecución es diferente para cada uno de ellos. Los test de nodo añaden recursos de ROS que los hacen mucho más costosos a nivel de ejecución que los test a nivel de código.

Vamos a pasar a describir dos ejemplos en nuestra aplicación de ROS, uno para cada caso.

Code-Level Unit Tests

Empezamos de lo particular a lo más general. Para los test a nivel de código, como ya se ha explicado anteriormente se utilizará la librería *unittest*.
La forma de testear mediante esta librería en ROS es la misma que para cualquier otra aplicación codificada en Python salvo por un pequeño cambio necesario para poder ver el resultado del test correctamente.
Vamos a testear la parte inicial de nuestra máquina de estados. Vamos a comprobar lo siguiente:

- Que se comprueba que los datos son válidos una vez estamos el primer estado.

Disponemos de una función que recibe los datos recibidos y comprueba si estos son válidos devolviendo un booleano como respuesta. Esta es la función que vamos a testear. Para ello le pasamos unos datos falsos para comprobar cuál es el resultado de la función *isInfoCorrect*.

```python
#!/usr/bin/env python
import unittest
from Utils import Utils

class UtilsTests(unittest.TestCase):

    def test_data_correct(self):
        mockData = {'name': 'Jinko', 'questions': 2, 'difficulty': 'facil'}
        outcome = Utils.isInfoCorrect(mockData)
        self.assertEquals(outcome, True)

if __name__ == '__main__':
    import rosunit
    rosunit.unitrun(PKG, 'UtilsTests', UtilsTests)
```

Figura 31: Testeo unitario en ROS

Las dos últimas líneas de nuestro test es donde realizamos la modificación que nos permite utilizar *unittest* con ROS. Se trata de envolver el test unitario con el paquete *rosunit* y ejecutarlo con la función *unitrun* para poder, así, producir el resultado de test correcto para ros. La imagen 25 muestra el resultado por consola al ejecutar los test.

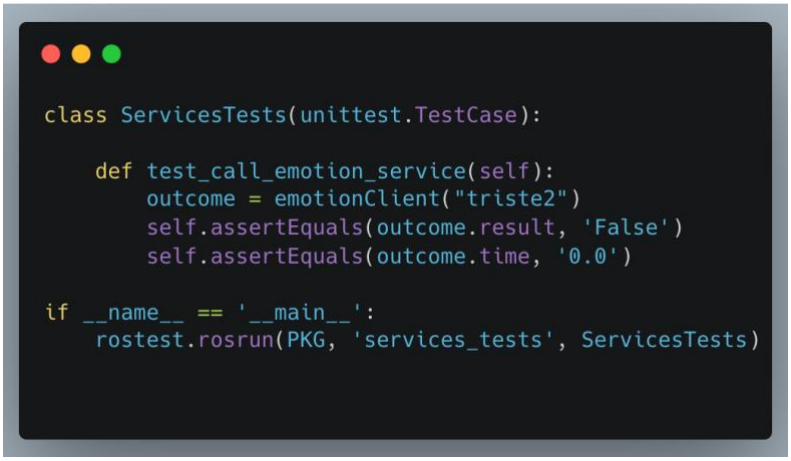
```
diana@MSIDiana:~/catkin_ws/src/action-template/src/test$ python -m utils
[ROSUNIT] Outputting test results to /home/diana/.ros/test_results/action_
[Testcase: test_data_correct] ... ok
[Testcase: test_save_data_correct] ... ok
----------------------------------------------------------------------
SUMMARY:
 * RESULT: SUCCESS
 * TESTS: 2
 * ERRORS: 0 []
 * FAILURES: 0 []
```

Figura 32: Resutlado tests unitarios

ROS-Nodel-Level Integration Tests

Vamos a pasar a los test a nivel de nodos, vamos a comprobar que el servicio de emoción es llamado correctamente, es decir, que la conexión se ha establecido entre estos dos nodos.

Para ello debemos importar la función encargada de llamar al ejercicio de emoción y ejecutar una llamada al mismo esperando que devuelva los datos esperados. De esta manera, si este test falla sabemos que es porque nuestro servidor de emociones no está activo o ha tenido algún problema ya que no recibe nuestra llamada y no obtenemos respuesta de él.



```
class ServicesTests(unittest.TestCase):

    def test_call_emotion_service(self):
        outcome = emotionClient("triste2")
        self.assertEquals(outcome.result, 'False')
        self.assertEquals(outcome.time, '0.0')

if __name__ == '__main__':
    rostest.rosrun(PKG, 'services_tests', ServicesTests)
```

Figura 33: Código del test test_call_emotion_service

El paquete necesita depender de rostest, por lo que habrá que añadir la dependencia de *rostest* en el archivo *package.xml.*

Nos encontramos con la misma circunstancia que en el caso anterior, el test debe ser ejecutado mediante *rostest*, así que debemos envolver nuevamente su ejecución mediante la función *rosrun*. Esta función tiene como condición que el test es un nodo y ejecutara todas las operaciones necesarias para que el nodo ejecute toda su lógica y después se destruya.

Si todo ha ido bien veremos por consola lo que aparece en la figura 26.

```
diana@MSIDiana:~/catkin_ws/src/action-template/src/test$ python -m emotion
[ROSUNIT] Outputting test results to /home/diana/.ros/test_results/action_template/rosunit-services_tests.xml
[Testcase: test_call_emotion_service] ... ok
----------------------------------------------------------------------
SUMMARY:
 * RESULT: SUCCESS
 * TESTS: 1
 * ERRORS: 0 []
 * FAILURES: 0 []

[ROSUNIT] Outputting test results to /home/diana/.ros/test_results/action_template/rosunit-services_tests.xml
```

Figura 34: Resultado test servicio emociones

Capítulo 8

RESULTADOS

Una vez terminado nuestro proyecto vamos a pasar a detallar el flujo de acción del psicólogo con la aplicación web y del niño con el robot.

Lo primero será iniciar sesión mediante email y contraseña. Esto nos dará acceso a la tabla de pacientes, desde aquí se podrán realizar dos acciones: Acceder a la información detalla de cada paciente y empezar los ejercicios.

Para empezar los ejercicios se especificará el número de preguntas para el ejercicio de emociones y la dificultad para el ejercicio de orientación. Esta información será introducida mediante un formulario. En la figura 35 se muestra el diagrama de flujo del proyecto con capturas de pantalla de todos estos pasos.

Al presionar el botón empezar, la aplicación mandará toda la información pertinente al servidor de ROS y el robot empezará con el saludo al niño.
Tras esto, le hará la primera pregunta. Esperará a que el niño levante la tarjeta como se muestra en la figura 34 y le dirá si es la correcta o no, si no lo es, repetirá la pregunta más despacio y si es la correcta pasará a la siguiente, y en el caso de que no queden más preguntas pasará al ejercicio de orientación.

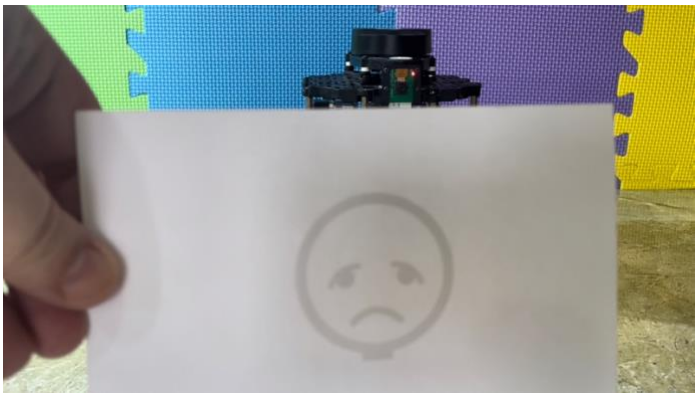


Figura 35: Detección de la tarjeta por parte del robot

En este ejercicio el robot le pedirá al niño que le siga y empezará a moverse. Llegado a su destino, el robot se despedirá del niño y acabará el ejercicio.

Una vez terminados los dos ejercicios los datos serán guardados en la base de datos, los aciertos del ejercicio de emociones y el tiempo empleado en ello.

De vuelta a la aplicación si se recarga la página se podrán ver las gráficas actualizadas y un formulario para que el psicólogo o terapeuta puede añadir comentarios y/o observaciones sobre el ejercicio de orientación o sobre la sesión en general.

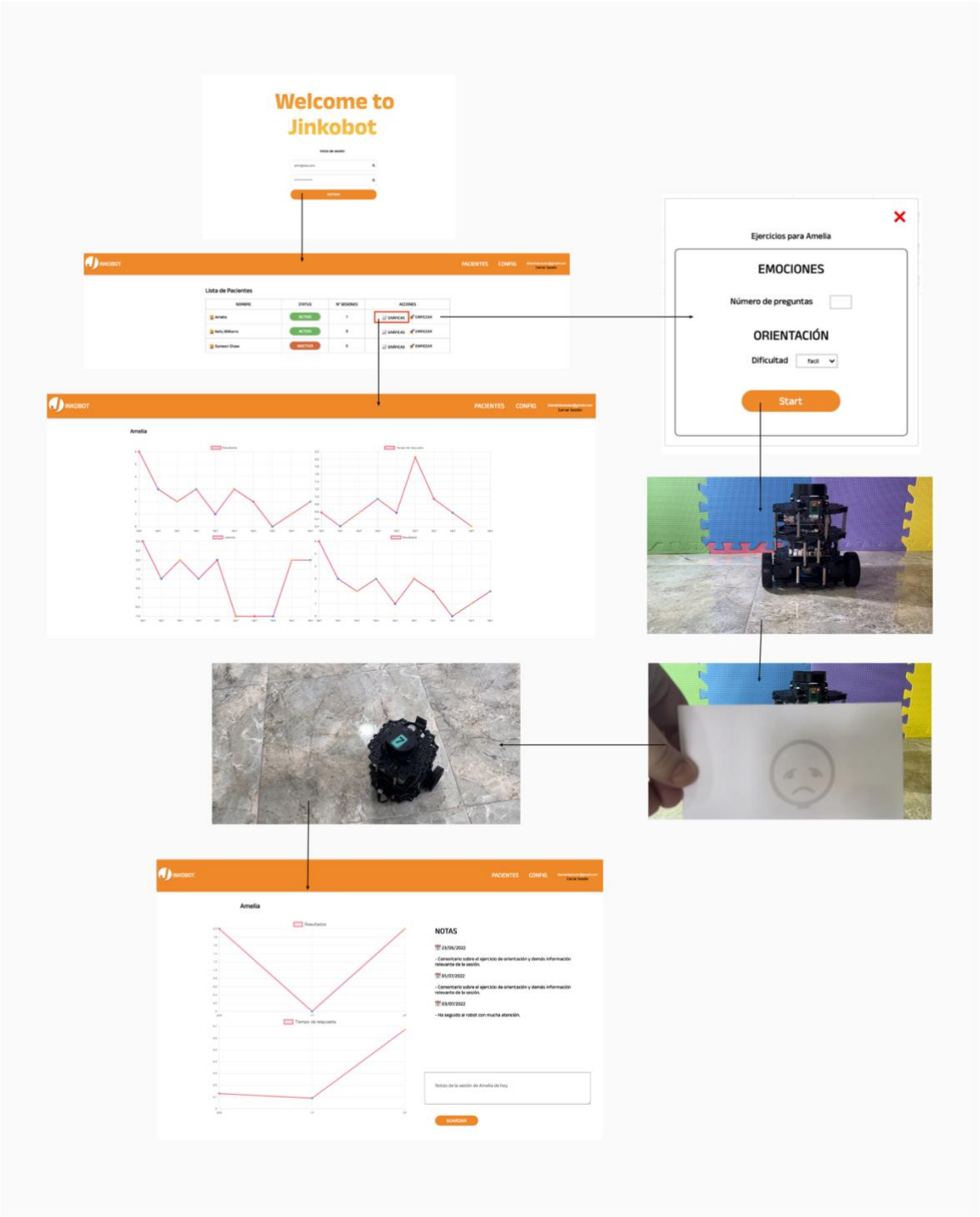


Figura 36: Diagrama de flujo del proyecto

En este enlace se puede ver un video donde se muestra todo el proceso, desde el login del psicólogo, hasta la visualización de resultados.

Demostración Robot Jinkobot

Capítulo 9

CONCLUSIONES

A lo largo de este proyecto, me he encontrado con diferentes fases, las iniciales, dónde creía todo posible, y la lista de funcionalidades a desarrollar era interminable, para luego encontrarme con la realidad, la de tener que priorizar y seleccionar lo necesario para presentar un buen trabajo final de grado con las horas asignadas para ello.

Y en estos primeros momentos, dónde todo es posible, se me ocurrieron mil y una cosas para añadir al proyecto, en aras de poder presentarlo al algún concurso o incluso ¿Por qué no? Llevarlo a la realidad. Paso a enumerar alguna de las ideas que por envergadura no se pudieron llevar a cabo, pero me gustaría poder añadirle en un futuro:

- Generación de rutinas por parte del psicólogo, pudiendo añadir preguntas, ejercicios y demás rutinas.
- Un semáforo con leds para decirle al niño cuando debe contestar.
- Añadirle una pantalla al robot para poder ver ilustraciones que acompañen a los ejercicios.
- Generación de informes.
- Apartado de relajación, con ejercicios de respiración, por si el niño se pone nervioso.
- Poder observar la atención del niño, saber si está mirando al robot o no.
- etc

Como proyecto para poner en práctica lo aprendido durante estos cuatro últimos años, pero, sobre todo, para encontrar la fusión perfecta entre lo tecnológico y lo social, Jinkobot me ha resultado la experiencia perfecta.
He podido investigar sobre un tema que desconocía, poniendo, ahora, en mucho más valor los problemas y las dificultades a las que se enfrentan las personas con diversidad funcional y de las que no somos apenas conscientes.

He podido trabajar en web, robótica, visión artificial y la nube. Y, aunque, pudiera parecer muchas tecnologías diferentes, ha sido un auténtico reto entender qué se quería hacer y como conseguirlo.

Este proyecto me ha permitido, también, poner un punto final espléndido, al menos desde mi sensación como alumna, a estos cuatro años. Tecnologías interactivas ha supuesto un antes y un después en mi vida.

Mi intención con la tecnología es la de ayudar a la gente, solucionar problemas, agilizar procesos… La tecnología por la tecnología, sin un uso detrás, para mi, se queda en un bonito museo cerrado al público.

BIBLIOGRAFÍA

[1] Tapus, A., Peca, A., Aly, A., A. Pop, C., Jisa, L., Pintea, S., Rusu, A. S., & David, D. O. (2012). *Children with autism social engagement in interaction with Nao, an imitative robot. Interaction Studies. Social Behaviour and Communication in Biological and Artificial Systems*, 13(3), 315–347. https://doi.org/10.1075/is.13.3.01tap

[2] Bharatharaj, J., Huang, L., Mohan, R. E., Al-Jumaily, A., & Krägeloh, C. (2017). Robot-assisted therapy for learning and social interaction of children with autism spectrum disorder. *Robotics*, *6*(1). https://doi.org/10.3390/robotics6010004

[3] Billard, A., Robins, B., Dautenhahn, K., & Nadel, J. (2007). Building robota, a mini-humanoid robot for the rehabilitation of children with autism. *Assistive Technology*, *19*(1), 37–49. https://doi.org/10.1080/10400435.2007.10131864

[4] CABRERA, Fausto R; RODRÍGUEZ, Cristian F; ZABALA, Mónica A y SANTACRUZ, Fabricio J. *Implementación de técnicas de visión artificial para el beneficio de niños con déficit de atención e hiperactividad a través de un ejercicio psicomotriz utilizando el robot NAO* [Artículo]. Disponible en internet en la dirección: http://www.revistaespacios.com/a19v40n32/19403203.html (Consultado en marzo 2020)

[5] Alonso, José R. *Realidad aumentada y autismo* [Artículo] Disponible en internet en la dirección https://jralonso.es/2018/06/04/realidad-aumentada-y-autismo/ (Consultado en marzo de 2020)

[6] Ciclo de gestión RAEE. Página web de Ministerio de trasformación ecológica y el reto demográfico. Disponible en internet en la dirección https://www.miteco.gob.es/es/calidad-y-evaluacion-ambiental/temas/prevencion-y-gestion-residuos/flujos/aparatos-electr/electricos-y-electronicos-ciclo-de-gestion.aspx (Consultado en mayo 2022)

[7] Paz Penagos, H.; Ortiz Niño, M. A.; Arévalo López, J. Aborde de Proyectos Integradores en Ingeniería Electrónica con Metodología CDIO. In *Proceedings of the 13th Latin American and Caribbean Conference for Engineering and Technology Engineering Education Facing the Grand Challenges What Are We Doing?*; LACCEI, 2015. https://doi.org/10.18687/LACCEI2015.1.1.154.

[8] (SLAM) Navigating While Mapping https://navigation.ros.org/tutorials/docs/navigation2_with_slam.html (Consultado en diciembre 2022)

[9] Documentación oficial de RVIZ http://wiki.ros.org/rviz (Consultado en diciembre 2022)

[10] Documentación oficial de Jest https://jestjs.io/es-ES/ (Consultado en noviembre 2021)

[11] Documentación oficial de React Testing Library https://testing-library.com/docs/react-testing-library/intro/ (Consultado en noviembre 2021)