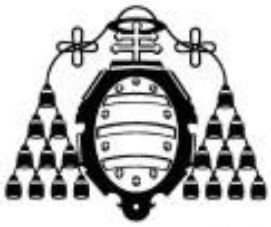
UNIVERSIDAD DE OVIEDO



ESCUELA DE INGENIERÍA INFORMÁTICA

PROYECTO FIN DE MÁSTER

"INVESTIGACIÓN SOBRE LA ADAPTACIÓN DE ROBOTS PARA FUNCIONES DE INTÉRPRETE"

DIRECTOR: Álvarez Gutierrez, Dario

CODIRECTOR: Sánchez González, Ignacio

AUTOR: Suárez Martínez, Daniel

Vº Bº del Director del Proyecto

Agradecimientos

A mis padres, a mi hermana, a mi cuñado y a mi sobrina, por todo el apoyo que me han brindado.

A mis amigos por animarme con él y a mis compañeros del máster por ser buenos compañeros durante estos dos cursos.

A los profesores Ignacio y Darío por su estímulo y constante ayuda.

Resumen

Este proyecto consiste en la investigación y estudio del estado del arte de la robótica así como el diseño y desarrollo de diferentes aplicaciones que dotan a un robot de funciones que le permiten ejercer de intérprete de distintas maneras, además de que estas funciones puedan tener efecto de forma remota, lo cual aborda la utilización de **ingeniería web** en este proyecto.

En el presente documento se explica en qué es, en qué consiste y cómo se utiliza el Robot Operating System (ROS), qué recursos se han utilizado, cómo se ha desarrollado este proyecto y la exposición de los fallos, problemas y limitaciones encontrados durante el mismo.

En este proyecto se han creado aplicaciones combinables entre sí que permiten a los robots establecer comunicaciones remotas con otros robots o a través de una página web en la cual también es posible mostrar la traducción de la información recibida y traducir la información enviada de vuelta al robot. También se muestra cómo hacer que el robot lea mediante voz la información que se le ha enviado desde el otro punto de la comunicación, además de ser capaz de reconocer comandos de voz para poder enviarlos. Por último, también se ha investigado y desarrollado, con algunas limitaciones, el uso de robots para la interpretación de lenguaje de signos mediante una mano robótica, ampliando las funciones de intérprete del robot, la cual también puede combinarse con las funciones anteriormente mencionadas, pudiendo usar la mano de forma remota y también a través de comandos de voz.

Palabras Clave

Robot, Web, Remoto, Intérprete, Robot Operative System, ROS, Lengua de signos, Lenguaje de signos, Reconocimiento de voz, Comandos de voz, Habla a texto, Texto a habla, Mano robótica.

Abstract

This project contains the investigation and study of the state of art of robotics and the design and development of different applications that give to a robot functionalities that allow it work as an interpreter in different ways, and this functions can have effect remotely, using the Web Engineering in this project.

In this document it's the definition and the explanation of what is and how to manage the Robot Operating System (ROS), what resources were used, how the project was developed and the explanation of errors, problems and limitations found during the development.

In the development of this project many applications were created and they can be combined between them, allowing the robots communicate remotely with another robots or with a web page were it is also possible show the translation of the information and translate the information before sending it to the robot. It is also how the robot can communicate with voice with the information it receive from the web page and how it can transform the voice he can hear into new messages for the web page. Part of this investigation are the development, with limitations, of the use of robots for sign language using a robotic hand, giving it more functions as an interpreter, using the hand remotely and also with voice.

Keywords

Robot, Web, Remote, Interpreter, Robot Operative System, ROS, Sign language, Speech to text, Text to speech, Voice recognition, Voice commands, Robotic hand.

Índice General

CAPÍTULO 1. MEMORIA DEL PROYECTO 21

1.1 Resumen de la Motivación, Objetivos y Alcance del Proyecto 21
1.2 Resumen de Todos los Aspectos 22

CAPÍTULO 2. INTRODUCCIÓN 23

2.1 Justificación del Proyecto 23
2.2 Objetivos del Proyecto 23
2.2.1 Objetivos principales 23
2.2.2 Desarrollo de los objetivos 23
2.3 Estudio de la Situación Actual 25
2.3.1 Evaluación de Alternativas 26
2.3.2 Funciones desarrolladas en este proyecto 27

CAPÍTULO 3. ASPECTOS TEÓRICOS 29

3.1 Robot Operating System (ROS) 29
3.2 Shadow Dexterous Hand ™ del Shadow Robot 29
3.3 Ubuntu 29
3.4 Virtualización 29

CAPÍTULO 4. PLANIFICACIÓN DEL PROYECTO Y RESUMEN DE PRESUPUESTOS 31

4.1 Planificación 31
4.2 Resumen del Presupuesto 33

CAPÍTULO 5. ANÁLISIS 35

5.1 Definición del Sistema 35
5.1.1 Determinación del Alcance del Sistema 35
5.2 Requisitos del Sistema 35
5.2.1 Obtención de los Requisitos del Sistema 35
5.2.2 Identificación de Actores del Sistema 36
5.2.3 Especificación de Casos de Uso 37
5.3 Identificación de los Subsistemas en la Fase de Análisis 40
5.3.1 Descripción de los Subsistemas 40
5.3.2 Descripción de los Interfaces entre Subsistemas 41
5.4 Diagrama de Clases Preliminar del Análisis 41
5.4.1 Diagrama de Clases 41
5.4.2 Descripción de las Clases 44
5.5 Análisis de Casos de Uso y Escenarios 47
5.5.1 Hacer hablar al robot de forma remota 47
5.5.2 Recibir la escucha del robot 49
5.5.3 Control de la mano robótica de forma remota 50
5.5.4 Control de la mano robótica por voz 52
5.6 Análisis de Interfaces de Usuario 52
5.6.1 Descripción de la Interfaz 53
5.6.2 Descripción del Comportamiento de la Interfaz 54
5.6.3 Diagrama de Navegabilidad 56
5.7 Especificación del Plan de Pruebas 57

5.7.1 Pruebas unitarias 57
5.7.2 Pruebas de Integración 58
5.7.3 Pruebas del sistema 58

CAPÍTULO 6. DISEÑO DEL SISTEMA 61

6.1 ARQUITECTURA DEL SISTEMA 61
6.1.1 Diagramas de Paquetes 61
6.1.2 Diagramas de Componentes 63
6.1.3 Diagramas de Despliegue 64
6.2 DIAGRAMAS DE INTERACCIÓN Y ESTADOS 65
6.2.1 Hacer hablar al robot de forma remota 65
6.2.2 Recibir la escucha del robot 66
6.2.3 Control de la mano robótica de forma remota 66
6.2.4 Control de la mano robótica por voz 67
6.3 DIAGRAMAS DE ACTIVIDADES 67
6.4 DISEÑO DE LA INTERFAZ 69
6.4.1 Interfaz web 69
6.4.2 Interfaz del robot 70
6.5 ESPECIFICACIÓN TÉCNICA DEL PLAN DE PRUEBAS 71
6.5.1 Pruebas Unitarias 71
6.5.2 Pruebas de Integración y del Sistema 73
6.5.3 Pruebas del sistema 74
6.5.4 Pruebas de Usabilidad y Accesibilidad 75
6.5.5 Pruebas de Rendimiento 78

CAPÍTULO 7. IMPLEMENTACIÓN DEL SISTEMA 81

7.1 ESTÁNDARES Y NORMAS SEGUIDOS 81
7.2 LENGUAJES DE PROGRAMACIÓN 81
7.3 HERRAMIENTAS Y PROGRAMAS USADOS PARA EL DESARROLLO 82
7.3.1 Robot Operating System (ROS) 82
7.3.2 Oracle VM VirtualBox 83
7.3.3 Gedit 83
7.4 CREACIÓN DEL SISTEMA 83
7.4.1 Problemas Encontrados 83
7.4.2 Descripción Detallada de las Clases 85

CAPÍTULO 8. DESARROLLO DE LAS PRUEBAS 89

8.1 PRUEBAS UNITARIAS 89
8.1.1 Síntesis de voz 89
8.1.2 Movimiento de la Shadow Hand (mano robótica) 91
8.1.3 Reconocimiento de voz (Pocketsphinx) 93
8.2 PRUEBAS DE INTEGRACIÓN Y DEL SISTEMA 95
8.3 PRUEBAS DE RENDIMIENTO 98

CAPÍTULO 9. MANUALES DEL SISTEMA 101

9.1 MANUAL DE INSTALACIÓN 101
9.1.1 Preparación del sistema 101
9.1.2 Preparación del proyecto 103
9.1.3 Preparación de las páginas web 104
9.2 MANUAL DE EJECUCIÓN 104
9.2.1 Iniciar la comunicación entre la página web y el robot 104
9.2.2 Iniciar el sonido 104

9.2.3 Iniciar el nodo de habla 105
9.2.4 Iniciar el reconocimiento de voz 105
9.2.5 Iniciar el nodo de control de la Shadow Hand 105
9.2.6 Iniciar la simulación de la Shadow Hand 106
9.3 MANUAL DE USUARIO 106
9.3.1 Instrucciones para el uso de la página web 106
9.3.2 Instrucciones para el uso del robot 107
9.4 MANUAL DEL PROGRAMADOR 107
9.4.1 Cambiar el diccionario del reconocimiento de voz 107
9.4.2 Cambiar el servicio de traducción 108
9.4.3 Modificar el manejo de la Shadow Hand 109

CAPÍTULO 10. CONCLUSIONES Y AMPLIACIONES 111

10.1 CONCLUSIONES 111
10.2 AMPLIACIONES 124

CAPÍTULO 11. PRESUPUESTO 125

CAPÍTULO 12. REFERENCIAS BIBLIOGRÁFICAS 127

12.1 LIBROS Y ARTÍCULOS 127
12.2 REFERENCIAS EN INTERNET 128

CAPÍTULO 13. APÉNDICES 129

13.1 GLOSARIO Y DICCIONARIO DE DATOS 129
13.2 CONTENIDO ENTREGADO EN EL CD-ROM 130
13.2.1 Contenidos 130
13.2.2 Código Ejecutable 131
13.2.3 Ficheros de Configuración 131
13.3 ÍNDICE ALFABÉTICO 132
13.4 CÓDIGO FUENTE 133
13.4.1 Nodos para el habla del robot 133
13.4.2 Nodos para el control de la Shadow Hand 134
13.4.3 Páginas web 146

Índice de Figuras

Figura 4.1 Diagrama de Gantt....32
Figura 5.2.3.1 Caso de Uso 1....37
Figura 5.2.3.2 Caso de Uso 2....38
Figura 5.2.3.3 Caso de Uso 3....39
Figura 5.2.3.4 Caso de Uso 4....40
Figura 5.4.1.1 Diagrama de clases de todas las comunicaciones....42
Figura 5.4.1.2 Diagrama de clases texto a voz y voz a texto....43
Figura 5.4.1.3 Diagrama de clases de texto a lengua de signos....43
Figura 5.4.1.4 Diagrama de clases de voz a lengua de signos....44
Figura 5.4.2.2 Mapa de engranajes de la Shadow Hand....46
Figura 5.5.1 Diagrama del habla del robot....47
Figura 5.5.1.1 Diagrama del habla del robot con traducción del mensaje....48
Figura 5.5.2 Diagrama del reconocimiento de voz del robot....49
Figura 5.5.2.1 Diagrama del reconocimiento de voz del robot con traducción....50
Figura 5.5.3 Diagrama de control de la mano robótica....50
Figura 5.5.3.1 Diagrama de control de la mano robótica con traducción de los mensajes....51
Figura 5.5.4 Diagrama de control de la mano robótica por voz....52
Figura 5.6.1.1 Boceto de una interfaz....53
Figura 5.6.1.4 Simulación de la Shadow Hand en Gazeboo....54
Figura 5.6.3 Diagrama de Navegabilidad....56
Figura 6.1.1 Diagrama de Paquetes....61
Figura 6.1.2 Diagrama de Componentes....63
Figura 6.1.3 Arquitectura del sistema....64
Figura 6.2.1 Diagrama de Interacción sobre el control remoto del habla del robot....65
Figura 6.2.2 Diagrama de Interacción sobre la escucha del robot....66
Figura 6.2.3 Diagrama de Interacción sobre el control de la mano robótica....66
Figura 6.2.4 Diagrama de Interacción sobre el control por voz de la mano robótica....67
Figura 6.3 Diagrama de actividad genérico de los nodos....67
Figura 6.3.1 Diagrama de actividad del sistema completo....69
Figura 6.4.1 Muestra de la interfaz de la web....69
Figura 6.4.2 Posiciones de la Shadow Hand al representar la V....70
Figura 6.4.2.1 Posiciones de la Shadow Hand al representar la L....70
Figura x.x Engranajes de la Shadow Hand y sus limitaciones....91
Figura x.x Prueba de rendimiento del traductor....98
Figura x.x Cómo aparece la IP en Linux....104

Capítulo 1. Memoria del Proyecto

1.1 Resumen de la Motivación, Objetivos y Alcance del Proyecto

Este proyecto consiste en estudiar la tecnología existente sobre robótica relacionada fundamentalmente con las comunicaciones entre dispositivos, entre los cuales se hayan dispositivos informáticos y robóticos, y las posibles interacciones entre máquinas y humanos para así poder desarrollar programas, que son patrones de órdenes, que permitan a dos personas establecer una comunicación usando a los robots como intermediarios que se encargarán de funciones de accesibilidad, permitiendo superar barreras tanto debidas a una discapacidad, como a la separación física entre las personas que se comunican o, también, al desconocimiento del idioma, pudiendo resolver varias de estas barreras a la vez.

Esta situación está resuelta con dispositivos portátiles, pero no con robots, los cuales disponen de funciones y posibilidades que no tienen estos dispositivos, ya que los robots se pueden mover allí donde se necesitan, pueden acompañar a una persona y también tienen partes mecánicas que en muchos casos buscan asemejar los robots a los humanos, es por ello que se suma a las posibilidades usar manos robóticas en apariencia similares a las humanas para interpretar lenguaje de signos, lo cual es útil para comunicarse por lenguaje de signos si una persona no puede escuchar al robot y no puede leer el texto de una pantalla del robot ya sea por distancia o porque el robot no disponga de dicha pantalla.

Este proyecto se centra en averiguar cómo se pueden lograr estos objetivos y qué hace falta para poder mejorar o completar alguno de los aspectos investigados.

1.2 Resumen de Todos los Aspectos

Este proyecto consiste en la investigación de cómo desarrollar funcionalidades de intérprete para un robot y combinarlas con la ingeniería web para hacer uso de ellas.

El robot establecerá una comunicación entre dos personas, convirtiendo la voz de una en texto que se mostrará en una página web y el texto de la página web se convertirá en voz que para comunicarse con el usuario en frente del robot. Esto permite, por ejemplo, comunicarse con voz con una persona con problemas de audición o habla que es quien estaría manejando la página web en este caso.

Otro caso de uso es mediante la página web o mediante voz controlar una mano robótica para realizar lenguaje de signos.

Capítulo 2. Introducción

2.1 Justificación del Proyecto

Este proyecto se desarrolla para cubrir la necesidad de distintos tipos de intérpretes para distintas situaciones, siendo los robots los que cubrirían esta necesidad, pudiendo adaptarse a la situación según lo requiera.

Los robots podrán pasar mensajes de texto a voz, mensajes de voz a texto, convertir cualquier mensaje en una interpretación de signos con un brazo robótico y traducir cualquier mensaje a otro idioma en el proceso.

2.2 Objetivos del Proyecto

2.2.1 Objetivos principales

1. Habla del robot por voz.
2. Comunicación entre el robot y una interfaz.
3. Traducción entre la comunicación del robot y la interfaz.
4. Reconocimiento de voz por el robot.
5. Investigación sobre el uso del lenguaje de signos con una mano robótica.

2.2.2 Desarrollo de los objetivos

Estos son los pasos para alcanzar cada objetivo del proyecto, los cuales se repiten de forma cíclica a la hora de desarrollar cada una de las funciones del robot:

1. Investigación de la tecnología existente.
    a. Búsqueda de información.
    b. Estudio de las opciones encontradas.
2. Experimentación con las opciones encontradas.
    a. Realización de tutoriales, en caso de que haya.
    b. Prueba de la tecnología.

   c. Valoración de la tecnología.

3. Elección de la tecnología.
4. Creación de las aplicaciones.

   a. Desarrollo de los programas.

   b. Pruebas de funcionamiento.

   c. Pruebas de conectividad y colaboración con las otras aplicaciones.

2.3 Estudio de la Situación Actual

Es visible que desde la creación de la robótica a los robots se les ha ido dotando de nuevas funciones, algunas de las cuales pueden ser realizadas directamente por humanos y otras que no pueden ser llevadas a cabo por una persona, pero siempre con el objetivo de facilitarnos la vida o de entretenernos. Se encargan de labores repetitivas, costosas, peligrosas o difíciles de realizar por las personas.

Una de las investigaciones más importantes de la robótica es su aplicación en la medicina[1], siendo el área de mayor impacto la cirugía, pudiendo realizarse incluso de forma remota[2] cuando la calidad de las comunicaciones lo permitan, es decir, que haya poca latencia en las comunicaciones y estén implementados sistemas de seguridad ante la pérdida de información durante la transmisión ya que se requiere la realización de movimientos muy precisos al controlar el robot mientras se ve en tiempo real el curso de la operación. La robótica en medicina clave para las intervenciones invasivas mínimas, es decir, cuando se requiere para la intervención que las incisiones sean lo más pequeñas posible, reduciendo el riesgo de infección y el tiempo de recuperación del paciente. Aún así, el uso de estos robots requiere una preparación y unos costes que suponen su lenta implantación en los centros hospitalarios, pero aún así esta tecnología sigue avanzando y adaptándose para resolver mayores problemas.

Otra función de la robótica son el reconocimiento de voz[3] y la posibilidad de hablar[4], con la posibilidad de que un humano y un robot se comuniquen como dos personas, incluso pueden recibir un mensaje de una persona para comunicárselo a otra.

Algunos robots se diseñan para representar comportamientos ante estímulos, ya sean estas respuestas automáticas preestablecidas por su programación como manuales controladas de forma remota, incluyendo también sensores de presión. Estos robots son muy importantes para el estudio del comportamiento humano y del autismo[5] y otras patologías relacionadas, sirviendo a su vez como herramientas para estimular en niños autistas las habilidades sociales, como es el caso de Keepon, del que ya se comercializan versiones con menos funciones.

Se han creado robots para ejercer la función de avatares[6], es decir, establecer una comunicación entre un usuario y un operador del robot, el cual de forma remota lo controla, recibe información de los estímulos del robot y puede controlar el comportamiento de este, pudiendo ser el robot una interfaz entre el operador y el usuario, sirviendo solo para establecer una comunicación, o una "marioneta" del operador, interpretando a un personaje a través del robot, lo cual puede ser útil para fines educativos o de análisis o tratamiento de enfermedades psicológicas, como es el caso del autismo, en niños, sin que ellos piensen que las respuestas del robot provienen de un operador.

Actualmente se están diseñando y fabricando robots humanoides, robots que se parezcan y se muevan como personas y puedan desempeñar algunas de sus funciones, pero con mayor interés en hacerlos autónomos[7], es decir, que puedan moverse libremente en un entorno e interactuar en el mismo sin intervención humana, para lo cual requieren mayor batería, no ser controlados de forma remota y no requerir de procesamiento externo.

2.3.1 Evaluación de Alternativas

2.3.1.1 Sistemas de desarrollo robótico

Cuando se habla de robótica, los esfuerzos suelen centrarse en el desarrollo de componentes y de su integración en el sistema, pero esto no siempre supone que se cree un sistema o arquitectura que se pueda utilizar para facilitar el desarrollo y aprovechamiento de software para el robot.

Este fue uno de los propósitos de R.T. Pack, D. Mitchell Wilkes y K. Kawamura [8], cuyo propósito era crear una arquitectura que fuese funcional para una amplia variedad tanto de robots como de componentes y subsistemas de los mismos, así como flexible para ofrecer distintas alternativas en el desarrollo, usando un enfoque basado en objetos.

Otro ejemplo lo encontramos en OROCOS (Open Robot Control Software) [9], con licencia de código abierto, permitiendo su estudio para su uso y modificación, a la vez que permite a los creadores de nuevo código ser los propietarios del mismo. Sus objetivos son la creación de sistemas basados en componentes, permitiendo combinar partes de distintos desarrolladores, tanto los que ofrecen sus módulos de forma gratuita como de pago, y centrarse en el manejo en tiempo real de robots y otra maquinaria, pero permitiendo la comunicación con otros sistemas para combinar con otras funciones.

Finalmente tenemos el sistema ROS[10], basado en una estructura de nodos, los cuales se comunican entre sí para llevar a cabo las distintas funciones del robot, lo cual permite en centrarse en el desarrollo de estos componentes y tratarlos por separado si hubiera algún error o se necesitase implementar alguna mejora. Al igual que ocurre con OROCOS, ROS se basa en un desarrollo colaborativo, donde las creaciones de distintos desarrolladores se puedan combinar y utilizar en diversos sistemas. Más adelante en el presente documento se indagará más en qué consiste ROS y en cómo se utiliza para este proyecto.

2.3.1.2 Lengua de signos

Existen proyectos para la representación de manos virtuales que realizan mensajes de signos[11], ya que las manos robóticas no suelen diseñarse para este propósito, permitiendo por medio de una mano virtual, cuya forma y articulaciones están basadas en las de las manos reales, la interpretación de este lenguaje de signos.

En 1989 Oaktree Automation desarrolló una mano robótica para la comunicación por lengua de signos apoyada, es decir, lengua de signos para personas que carecen de visión y de oído[12].

En este proyecto se hace uso de la versión virtual de una mano robótica existente para la comunicación mediante lenguaje de signos.

2.3.1.3 Control remoto

En la actualidad el uso de sistemas informáticos de forma remota es algo común, y los robots no están exentos de esta posibilidad. Tenemos ejemplos como el de Xavier [13], un robot autónomo pero que permitía s través de una página web hacerle cambiar de habitación y enviar un correo electrónico cuando esta tarea finalizase, a la vez que desde otra página web enviaba imágenes de su situación.

Otro ejemplo lo encontramos en Minerva[14], un robot para guías por museo, que permite a través también de una página web controlarlo para visitar un museo y hacer un recorrido.

En este proyecto se utiliza una interfaz web tanto para controlar las funciones del robot como para recibir mensajes del mismo.

2.3.2 Funciones desarrolladas en este proyecto

Este proyecto pretende añadir a un robot la función de intermediario entre personas con distintas formas de comunicación. Actualmente para que una persona sordo-muda se comunique con otra persona estas tienen que recurrir al lenguaje de signos o escribirse mensajes, aunque también está la posibilidad de la persona sordo-muda de leer los labios y si solo es sorda puede hablar, pero estas dos habilidades no todos las tienen adquiridas, de modo que si la persona que puede hablar no conoce la lengua de signos necesita recurrir a un intérprete. Con este proyecto se pretende convertir al robot en intérprete, escuchará lo que diga la persona hablante, lo convertirá a texto que la persona sordo-muda podrá leer y también podrá convertir el texto escrito por esta persona en voz que escuchará la persona hablante.

2.3.2.1 Robot escucha y habla

El robot puede escuchar lo que una persona le dice y convertir la escucha en texto que se mostrará en una página web en un navegador web que puede estar tanto en el propio ordenador conectado al robot como en un ordenador remoto. A su vez esta página también permite enviar texto al robot el cual reproducirá mediante voz, dirigiéndose a la persona que habla al robot.

Esto permitiría a una persona sordo-muda o en un lugar remoto comunicarse mediante habla con otra persona frente al robot, siendo este intérprete entre los dos.

2.3.2.2 Robot traductor

El sistema y el mecanismo es el mismo que el del punto anterior, pero la diferencia está en que el texto de la página, tanto el entrante como saliente, es traducido por un servicio web externo a otro idioma, lo cual suma la cualidad de traducción en la comunicación.

2.3.2.3 Lenguaje de signos interpretados por el robot

En este proyecto se utiliza una mano robótica para realizar lenguaje de signos. Los mensajes que debe interpretar se pueden enviar tanto desde la página web como pueden ser los que se escuchen por voz, tan solo hay que especificar en el código de qué canal leerá el texto a interpretar. Se utiliza una versión simulada (virtual) de la Shadow Hand[15], la mano robótica del Shadow Robot[1].

[1] http://www.shadowrobot.com/

Capítulo 3. Aspectos Teóricos

3.1 Robot Operating System (ROS)

El Robot Operating System (ROS) es un sistema flexible para la creación de software para robots. Consiste en un conjunto de herramientas y librerías para simplificar la tarea de crear funciones para robots para diversas plataformas robóticas, ya que es una tarea de por sí compleja y difícil, permitiendo a más gente y equipos de investigación desarrollar este tipo de software.

ROS fue creado para promover el desarrollo colaborativo, de modo que las aplicaciones creadas por los distintos usuarios del sistema se puedan combinar entre sí, ayudándose los desarrolladores los unos a los otros con sus desarrollos.

3.2 Shadow Dexterous Hand ™ del Shadow Robot

La Shadow Dexterous Hand es una mano robótica muy similar a la humana para su uso en robots. El modelo real posee sensores de fuerza y de tacto, permitiendo realizar tareas lo más próximo posible a cómo lo haría una mano humana.

Hace uso de interfaces estandarizadas y puede ser usada en otros robots, así como adaptarse de forma personalizada según la necesidad.

En el proyecto se utiliza una simulación en Gazebo de esta mano.

3.3 Ubuntu

Ubuntu es un sistema operativo basado en Linux con una filosofía de software libre. La mayoría de los paquetes de los que se compone comparten esta filosofía, lo cual otorga a los usuarios la libertad de estudiar, adaptar, modificar y distribuir sus propias versiones. Ubuntu también permite utilizar software propietario.

3.4 Virtualización

En un principio este proyecto no consiste ni se centra en virtualización, pero debido a una serie de problemas de compatibilidad entre Ubuntu y el hardware utilizado, la virtualización permitió utilizar sobre ese hardware versiones compatibles de ROS, ya que durante el desarrollo de este proyecto la versión de ROS compatible con Ubuntu 14.04, Indigo Igloo, se retrasó y habría sido imposible finalizar el proyecto antes, de modo que se le hará una pequeña mención.

Capítulo 4. Planificación del Proyecto y Resumen de Presupuestos

4.1 Planificación

En la siguiente página se incluye el diagrama de Gantt de la planificación inicial del proyecto como fue planeado en un principio:

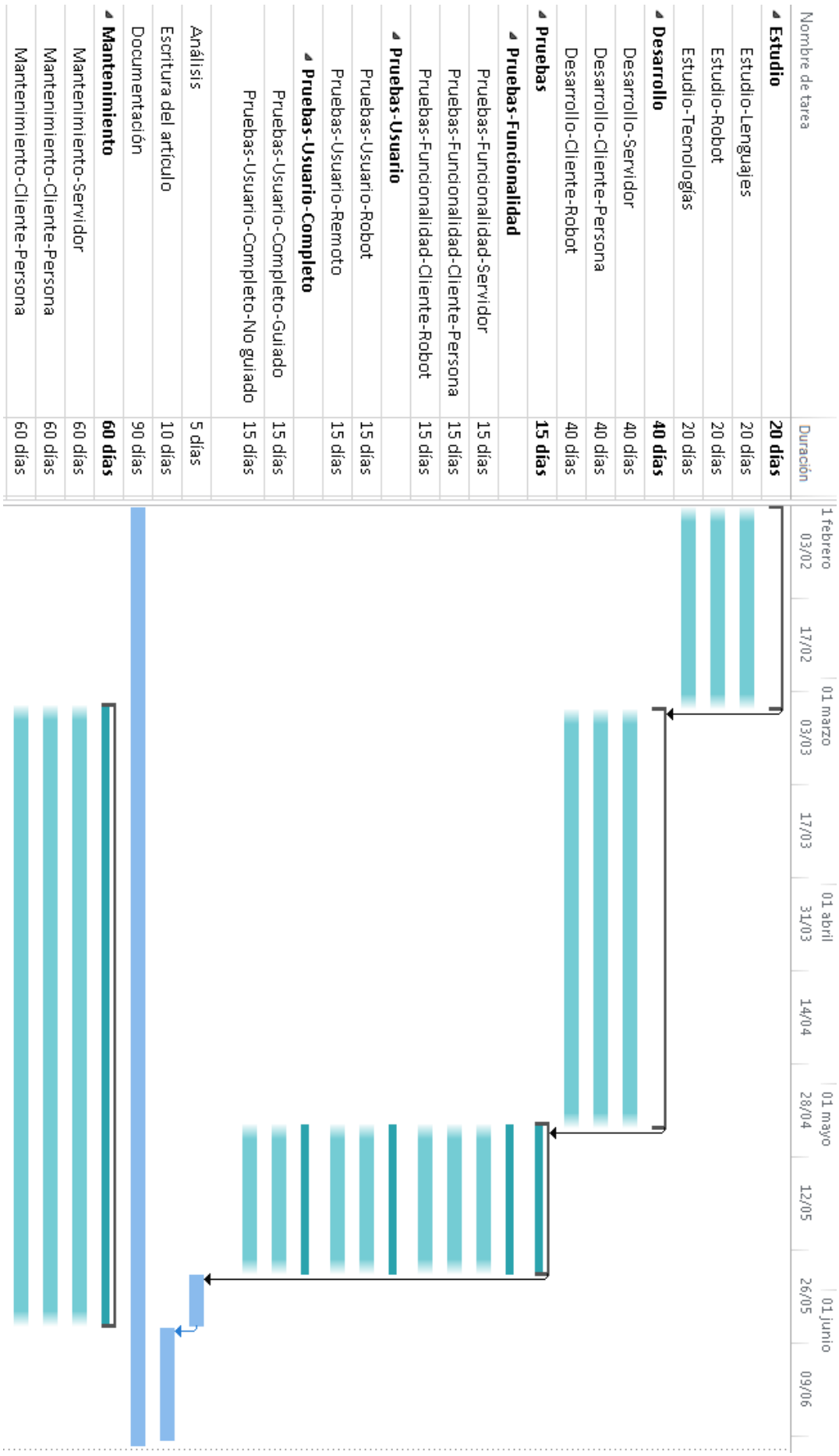


Figura 4.1 Diagrama de Gantt

4.2 Resumen del Presupuesto

Item	Concepto	Tiempo de desarrollo (meses)	Precio mensual	TOTAL
1	Preparación del material	0,07	1.500,00 €	105,00 €
2	*Investigación de las tecnologías*	2,50	1.500,00 €	3.750,00 €
3	*Desarrollo del prototipo*	1,50	1.500,00 €	2.250,00 €
4	*Documentación*	0,50	1.500,00 €	750,00 €
5	*Material y mantenimiento del mismo*	5,00	200,00 €	1.000,00 €
			Subtotal	7.855,00 €
			IVA (21%)	1.649,55 €

Capítulo 5. Análisis

5.1 Definición del Sistema

5.1.1 Determinación del Alcance del Sistema

Este proyecto consiste en el estudio del uso de la tecnología existente para dotar a robots de funciones de intérprete, existiendo una comunicación entre nodos que es posible entre máquinas separadas.

Las funciones a desarrollar son:

- Comunicación entre el robot y una página web, ofreciendo un servicio web de comunicación y control del robot.
- Envío de texto al robot para su conversión en voz.
- Reconocimiento de voz.
- Traducción del texto recibido del robot y del texto a enviar al robot a otros idiomas.
- Intentar implementar dentro de las posibilidades la representación de lenguaje de signos mediante el uso de una mano robótica.

5.2 Requisitos del Sistema

5.2.1 Obtención de los Requisitos del Sistema

Código	Nombre Requisito	Descripción del Requisito
R1.1	Instalación del sistema operativo	Debe estar instalado un sistema operativo que usaremos para desarrollar el proyecto.
R1.2	Instalación de máquinas virtuales	Se crearán máquinas virtuales para desarrollar y hacer pruebas del proyecto.
R1.3	Instalación de ROS en las máquinas virtuales.	Se instalará y configurará ROS, así como los módulos necesarios, en las máquinas virtuales.
R2.1	Creación de Nodos	Creación de nodos que se encargarán de las distintas funciones que realizará el robot.
R2.2	Comunicación entre nodos	Los nodos han de poder comunicarse entre sí e intercambiar información.
R2.3	Comunicación con servicio web	Desde un servicio web se ha de poder intercambiar información, tanto de envío como de recepción, con los nodos del robot.
R.3.1	Texto a voz	El texto recibido por uno de los nodos ha de poder convertirse a voz.
R.3.2	Voz en distintos idiomas	Pueden haber distintas versiones del nodo de habla

		para distintos idiomas, por lo menos en inglés (idioma por defecto) y castellano.
R4.1	Reconocimiento de voz	Uno de los nodos debe ser capaz de hacer reconocimiento de voz y enviar la información a otro nodo o al servicio web.
R5.1	Traducción para mensajes entrantes	El servicio web debe hacer uso de un servicio web de terceros para traducir los mensajes que le lleguen desde un nodo del robot.
R5.2	Traducción para mensajes salientes	El servicio web debe hacer uso de un servicio web de terceros para traducir los mensajes que se envíen hacia un nodo del robot.
R6.1	Simulación de mano robótica	Se debe poder usar una simulación de una mano robótica real.
R6.2	Lenguaje de signos con la mano robótica	La mano robótica debe poder representar, dentro de sus limitaciones, lenguaje de signos.

Código	Nombre Requisito	Descripción del Requisito
R1.1	Insertar Usuario	Se debe añadir un usuario al sistema una vez leídos y validados sus datos.
R1.2	Leer Datos Usuario	Deben pedirse los datos completos de un usuario del sistema

5.2.2 Identificación de Actores del Sistema

ACTOR	FUNCIÓN
Administrador	Se encarga de activar los nodos de ROS necesarios en el robot a utilizar según el propósito a tratar.
Usuario del sitio web	Utiliza la página web para recibir y enviar mensajes al robot.
Usuario frente al robot	Puede hablarle al robot y escuchar sus respuestas.
Núcleo (robot)	Nodo en el que se publican todos los mensajes enviados entre los nodos para su comunicación entre sí.
Nodo de sonido (robot)	Permite al robot reproducir y grabar audio.
Nodo de habla (robot)	Convierte en voz la información recibida para comunicarse con el usuario.
Nodo de escucha (robot)	Interpreta el habla del usuario para enviarlo a otros nodos.
Diccionario de	Diccionario utilizado por el nodo de escucha con la pronunciación y las

vocablos	palabras que puede reconocer. Este diccionario debe ser generado según los intereses para el uso del programa.
Nodo de comunicaciones externas (robot)	Permite la comunicación con otras aplicaciones o medios que no sean un nodo del robot, como un servicio web.
Nodo de control de la mano robótica (robot)	Permite controlar la simulación de la mano robótica con la información recibida de otros nodos.
Mano robótica	Simulación de la mano robótica que se controlará para que realice, dentro de sus posibilidades, lenguaje de signos.

5.2.3 Especificación de Casos de Uso

5.2.3.1 Caso 1

Nombre del Caso de Uso
Hacer hablar al robot
Descripción
Un usuario remoto desde la página web escribe el texto que quiere que el robot hable con voz.

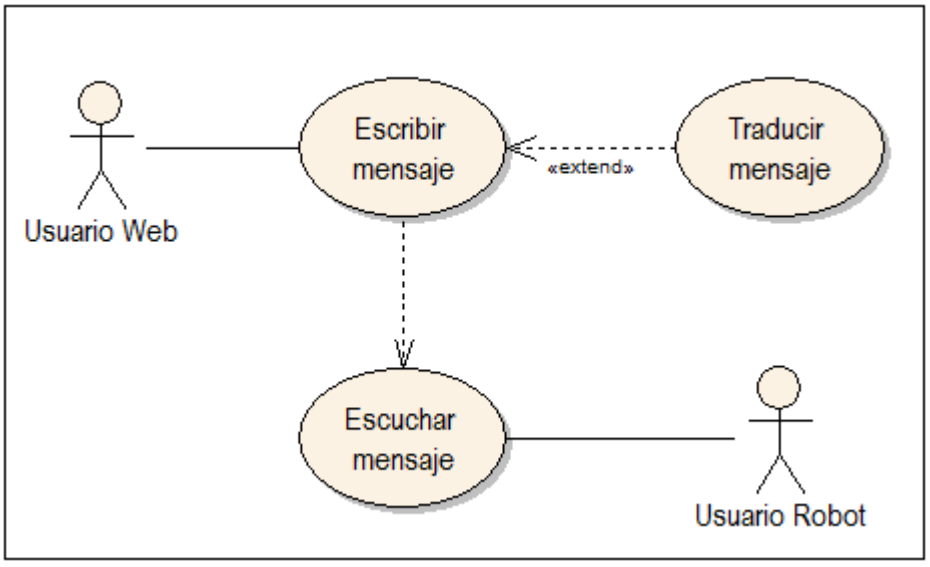


Figura 5.2.3.1 Caso de Uso 1

5.2.3.1.1 Variación del Caso 1

Nombre del Caso de Uso
Hacer hablar al robot un texto traducido
Descripción
Un usuario remoto desde la página web escribe el texto que quiere que el robot hable con voz y que será traducido antes de ser enviado al robot.

5.2.3.2 Caso 2

Nombre del Caso de Uso
Recibir mensajes del robot
Descripción
El robot escucha y envía convertido en texto su interpretación para que sea captado por el servicio web.

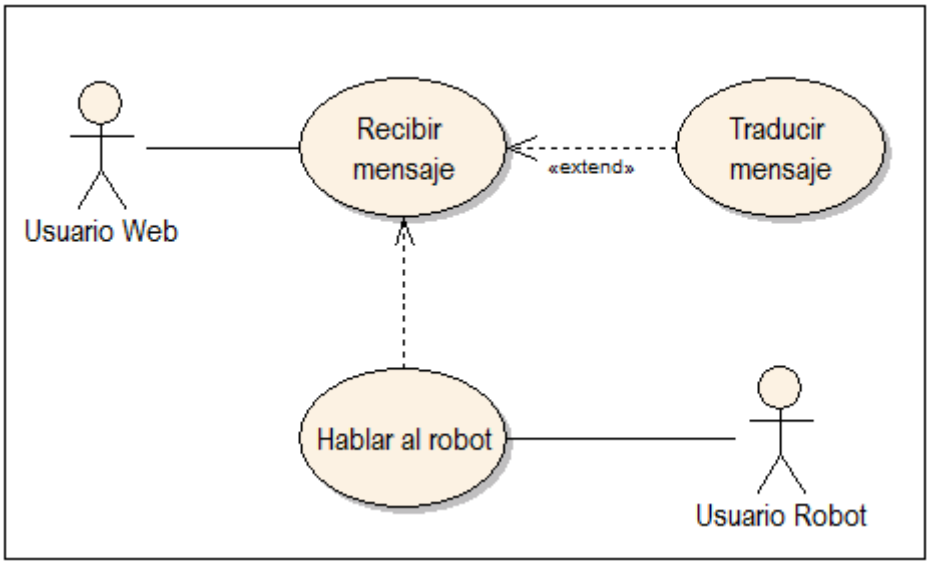


Figura 5.2.3.2 Caso de Uso 2

5.2.3.2.1 Variación del Caso 2

Nombre del Caso de Uso
Recibir mensajes traducidos del robot
Descripción
El robot escucha y envía convertido en texto su interpretación para que sea captado por el servicio web, el cual lo mostrará traducido.

5.2.3.3 Caso 3

Nombre del Caso de Uso
Convertir un mensaje a lenguaje de signos
Descripción

El usuario desde el servicio web enviará un mensaje, el cual el robot lo representará en lenguaje de signos con el simulador de la mano robótica.

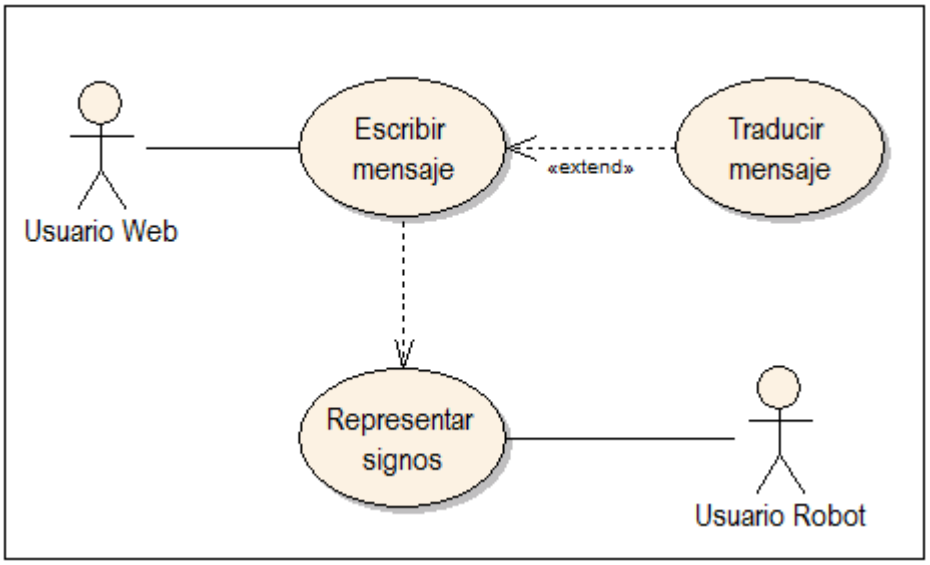


Figura 5.2.3.3 Caso de Uso 3

5.2.3.3.1 Variación del Caso 3

Nombre del Caso de Uso
Convertir un mensaje traducido a lenguaje de signos
Descripción
El usuario desde el servicio web enviará un mensaje previamente traducido, el cual el robot lo representará en lenguaje de signos con el simulador de la mano robótica.

5.2.3.4 Caso 4

Nombre del Caso de Uso
Convertir un mensaje de voz a lenguaje de signos
Descripción
El robot usará el reconocimiento de voz para representar el mensaje en lenguaje de signos con el simulador de la mano robótica.

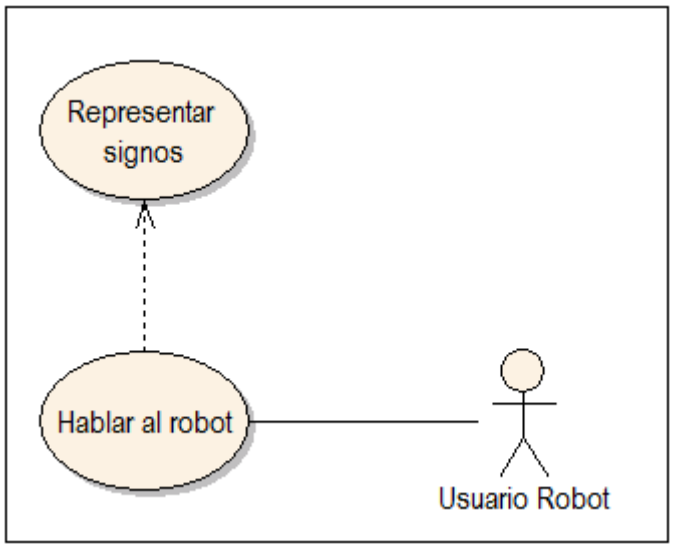


Figura 5.2.3.4 Caso de Uso 4

5.3 Identificación de los Subsistemas en la Fase de Análisis

5.3.1 Descripción de los Subsistemas

Al ser ROS un sistema operativo que permite la ejecución de nodos que se comunican entre sí por medio de un nodo central o núcleo (roscore), los actores que son precisamente nodos que se están ejecutando en el robot son los subsistemas de este proyecto.

Subsistema	Función
Núcleo (roscore)	Es el nodo central del sistema ROS y al que se dirigen todos los nodos. Existe la posibilidad de que estos nodos se estén ejecutando desde otra máquina y se dirijan a este núcleo.
Rosbridge	Es el nodo que se encarga de establecer comunicaciones entre el sistema y componentes que no pertenezcan a ROS, que pueden ser o no externos a la máquina donde se ejecuta el núcleo al que está conectado. En el caso de este proyecto, Rosbridge comunica el servicio web con el sistema.
Sound Play	Es el nodo encargado de que los nodos de ROS puedan reproducir sonidos. En este proyecto es usado para la síntesis de voz.
Nodo de habla	Convierte en voz el texto recibido.
Nodo de escucha (pocketsphinx)	Se encarga de interpretar el audio escuchado para convertirlo en texto. Puede utilizar distintos diccionarios según el contexto requerido.
Nodo de	Divide un texto recibido en letras para enviar las órdenes de movimiento a la

control de la mano robótica	mano robótica o su simulación para la interpretación de lenguaje de signos.

5.3.2 Descripción de los Interfaces entre Subsistemas

Los nodos de ROS se comunican entre sí pasándose mensajes, siendo estos una estructura de datos. Como estándar los tipos primitivos como números enteros o booleanos están soportados, así como cadenas de los mismos.

En ROS se utiliza un modelo de publicación y suscripción (publish / suscribe model), pero esto permite enviar mensajes en una sola dirección, con lo cual puede requerirse una respuesta a cada mensaje enviado para garantizar la comunicación en caso de ser necesario.

ROS permite que los nodos se ejecuten en distintas y que estos se conecten a un nodo nuclear al cual se referirán como maestro. No obstante, esto no será necesario en este proyecto.

Mediante el uso del nodo ROS Bridge, se pueden establecer comunicaciones tanto con otros nodos de ROS en ejecución en otras máquinas como con otros programas o aplicaciones que no se correspondan a ROS, permitiendo este nodo hacer de puente entre dicha aplicación y el núcleo, permitiendo enviar y recibir mensajes entre ellos.

5.4 Diagrama de Clases Preliminar del Análisis

5.4.1 Diagrama de Clases

5.4.1.1 Todas las comunicaciones posibles

En el siguiente diagrama de clases se muestran todas las comunicaciones posibles entre los distintos componentes del sistema, si bien es cierto que lo habitual es que se usen los componentes que se requieran según los intereses. Como se mencionó anteriormente, las comunicaciones son uno a uno y lo que se envían son mensajes en una estructura determinada para ser captados por los nodos que se suscriban al canal en el que se envían, de modo que no veo necesario hacer una representación más complicada de las comunicaciones.

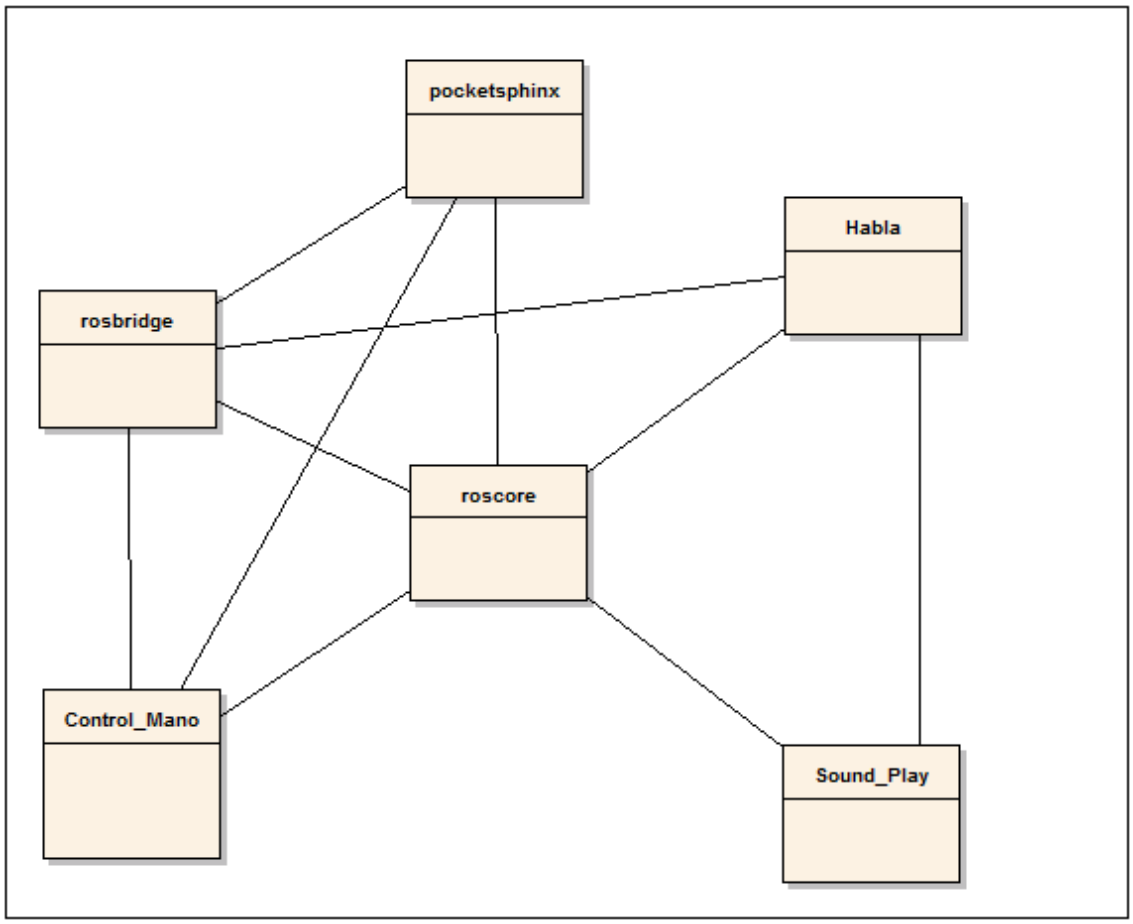


Figura 5.4.1.1 Diagrama de clases de todas las comunicaciones

Este diagrama también coincide con el que se usaría para hablar por voz al robot para mostrarlo en la página web y que este responda simultáneamente con lenguaje de signos y voz robótica.

5.4.1.2 De voz a texto en web y de texto en web a voz

Este es el caso en el que un usuario habla al robot y lo que este interpreta aparece en la página web, y a su vez el usuario que utiliza la página web escribe qué quiere que el robot le diga. Puede haber o no traducción en el proceso.

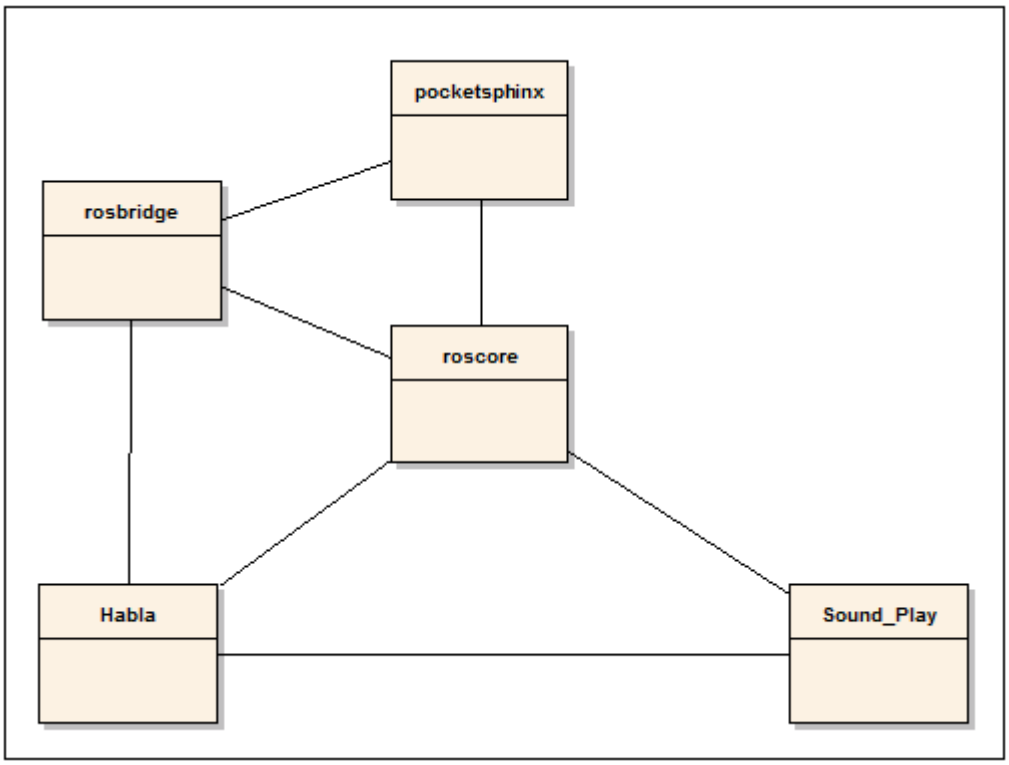


Figura 5.4.1.2 Diagrama de clases texto a voz y voz a texto

5.4.1.3 De texto a lengua de signos

La página web, por medio de rosbridge, se comunica con el nodo que permite mover la mano robótica o su simulación.

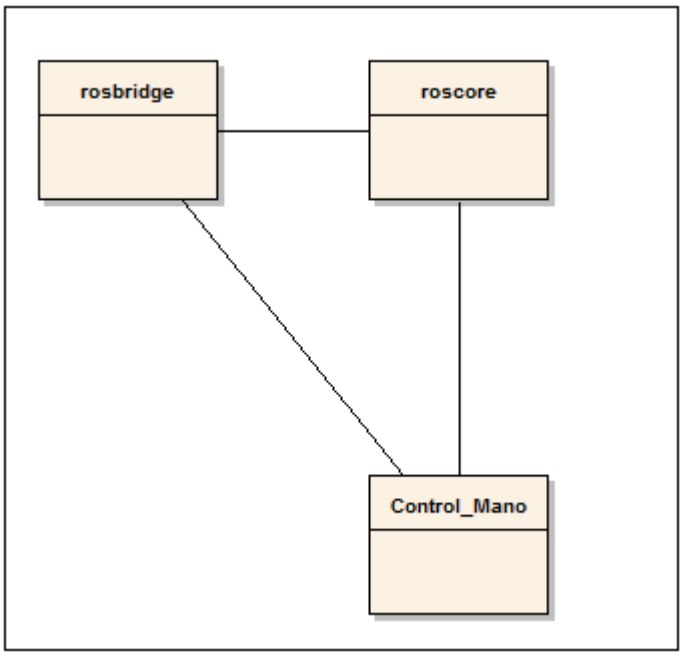


Figura 5.4.1.3 Diagrama de clases de texto a lengua de signos

5.4.1.4 De voz a lengua de signos

El nodo de control de mano se suscribe en vez de al canal en el que escribiría la página web al canal en el que escribe el reconocimiento de voz.

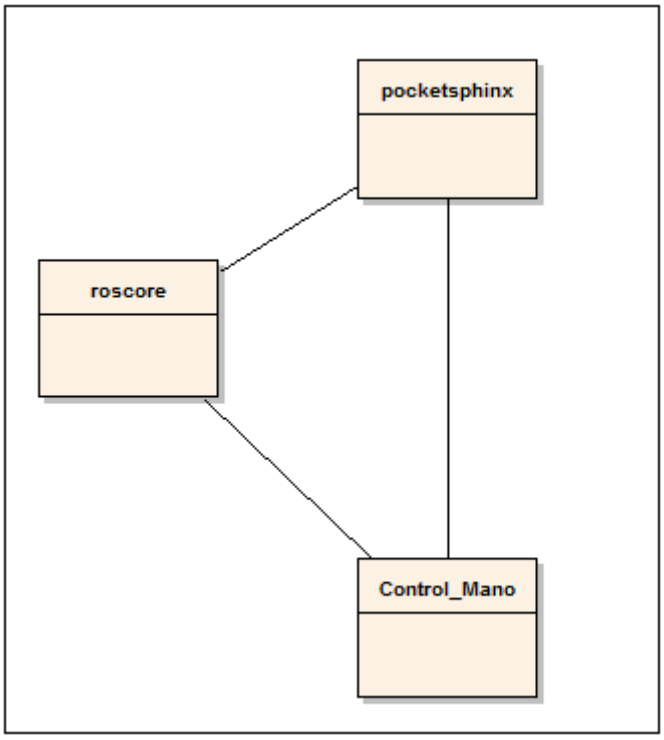


Figura 5.4.1.4 Diagrama de clases de voz a lengua de signos

5.4.2 Descripción de las Clases

En este apartado únicamente voy a describir las clases de los programas desarrollados para el proyecto y no las clases de las partes ya pertenecientes a ROS y usados en el mismo. Durante el desarrollo del proyecto se han creado más programas, pero solo han sido usado para pruebas y son en su gran parte variaciones de los programas de los tutoriales adaptadas para probar la funcionalidad de los programas importantes.

5.4.2.1 Clases de habla

Este programa tiene dos versiones con pocas variaciones en su código para modificar el idioma del habla para poder usar de forma independiente una versión en inglés y otra en castellano. Es solo una línea de código la que hace falta cambiar para usar una voz diferente que haya sido instalada previamente para hacer que el robot hable con otra entonación o en otro idioma.

La limitación de este programa está en que la síntesis de voz no reconoce los caracteres con tilde o la ñ, de modo que al usar la voz en castellano no dice estas palabras, sino el código de la letra.

Nombre de la Clase
escuchaWeb.py / escuchaWebEsp.py

Descripción
Es la clase que indica al robot qué debe decir y, en el caso de escuchaWebEsp, con qué voz ha de hacerlo.
Responsabilidades
Permanecer a la escucha de un canal hasta recoger un texto el cual será convertido en voz.
Atributos Propuestos
Ninguno
Métodos Propuestos
sleep: Duerme la aplicación el tiempo que se le ha pasado como atributo. **callback**: Cuando se recibe una cadena de texto se prepara para hablar y habla. **listener**: Inicia el nodo y se suscribe al canal de escucha para luego invocar el método callback y repetir el proceso. **main**: ejecuta el método listener la primera vez.

5.4.2.2 Clases de control de la mano.

La mano robótica utilizada es la Shadow Hand, de Shadow Robot, capaz de colocar los dedos en la posición necesaria para hacer los signos del lenguaje de signos correspondientes a una sola mano, pero con la limitación del movimiento del brazo y de la muñeca, de modo que la posición correcta de la mano no es posible. Este trabajo es de investigación, y esto demuestra que tenemos la tecnología necesaria para lograr este objetivo por completo, tan solo nos queda solucionar la limitación de la muñeca.

Para comprender un poco mejor el funcionamiento de esta clase se adjunta un gráfico del mapa de los engranajes de la mano robótica usada:

Nombre de la Clase
ControlSH.py / ControlSHVoz.py
Descripción
Este programa tiene métodos preparados para representar las letras en lenguaje de signos.
Responsabilidades
Permanece a la escucha del canal en el que escribe la web o el pocketsphinx y cuando recibe datos comienza la secuencia de representación
Atributos Propuestos
controller_type: Indica el tipo de control. **pubSass**: Valor de posición del brazo. **pubFfj3**: Valor de posición de la articulación 3 del dedo índice. **pubFfj0**: Valor de posición de las articulaciones 1 y 2 del dedo índice. **pubFfj4**: Valor de posición de la articulación 4 del dedo índice. **pubMfj4**: Valor de posición de la articulación 4 del dedo corazón. **pubMfj3**: Valor de posición de la articulación 3 del dedo corazón. **pubMfj0**: Valor de posición de las articulaciones 1 y 2 del dedo corazón. **pubRfj4**: Valor de posición de la articulación 4 del dedo anular. **pubRfj3**: Valor de posición de la articulación 3 del dedo anular. **pubRfj0**: Valor de posición de las articulaciones 1 y 2 del dedo anular. **pubLfj5**: Valor de posición de la articulación 5 del dedo meñique. **pubLfj4**: Valor de posición de la articulación 4 del dedo meñique. **pubLfj3**: Valor de posición de la articulación 3 del dedo meñique. **pubLfj0**: Valor de posición de las articulaciones 1 y 2 del dedo meñique. **pubThj5**: Valor de posición de la articulación 5 del dedo pulgar. **pubThj4**: Valor de posición de la articulación 4 del dedo pulgar.

pubThj3: Valor de posición de la articulación 3 del dedo pulgar. **pubThj2**: Valor de posición de la articulación 2 del dedo pulgar. **pubThj1**: Valor de posición de la articulación 1 del dedo pulgar. **tiempo_mucho**: Valor numérico de lo que deben durar las paradas largas. **tiempo_medio**: Valor numérico de lo que deben durar las paradas de duración media. **tiempo_poco**: Valor numérico de lo que deben durar las paradas cortas. **tiempo_poquisimo**: Valor numérico de lo que deben durar las paradas más cortas.
Métodos Propuestos
cierra: Cierra la mano con el pulgar dentro. **abre**: Abre la mano, esperando un rato antes de mover el pulgar. **abre2**: Abre la mano empezando por el pulgar antes de mover el resto de dedos. **abre3**: Abre la mano moviendo todos los dedos a la vez. **a**: Pone los dedos en la posición para representar la a. **b**: Pone los dedos en la posición para representar la b. **c**: Pone los dedos en la posición para representar la c. ***[...]*** **z**: Pone los dedos en la posición para representar la z. **mover**: A partir de la palabra recibida empieza la secuencia de movimiento de la mano según las letras a representar. **listener**: Permanece a la escucha y en caso de recibir datos ejecuta el método escucha. **main**: Inicia listener la primera vez.

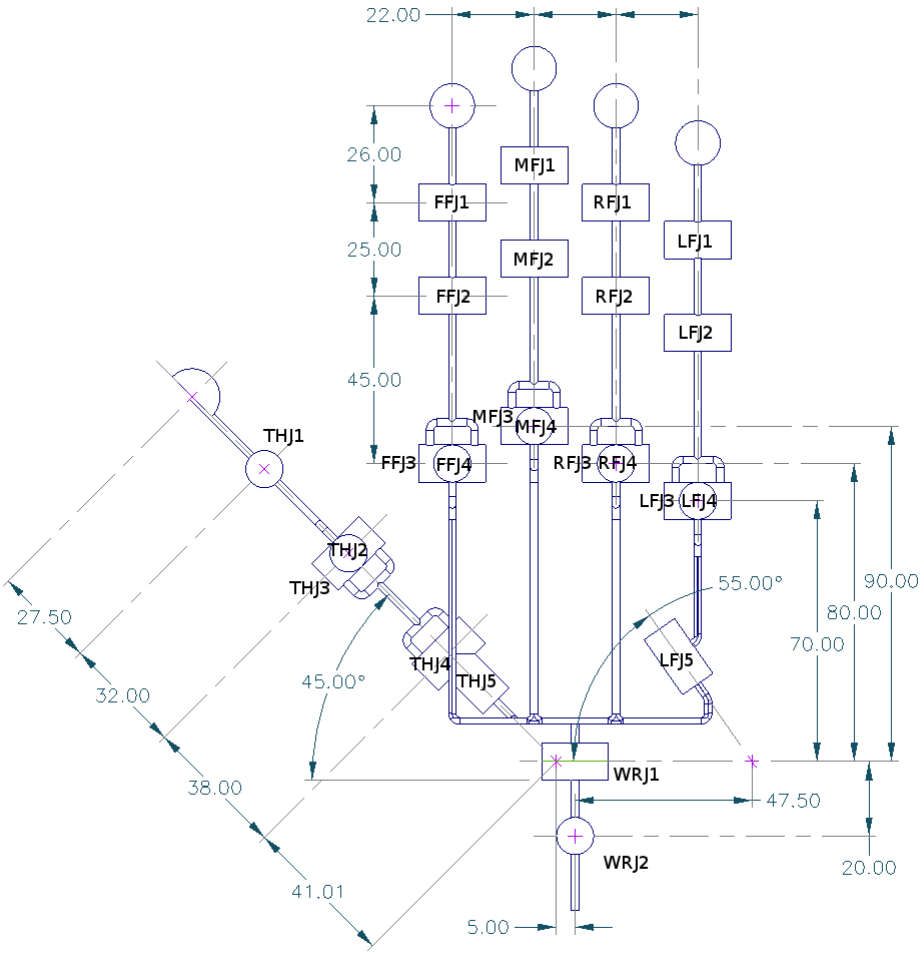


Figura 5.4.2.2 Mapa de engranajes de la Shadow Hand

5.4.2.3 Página web

Nombre de la Clase
saludа.html / saludaTraduce.html / greet.html
Descripción
Es la página web que se comunicará con el robot, intercambiando mensajes con él.
Responsabilidades
Intercambiar mensajes con el robot, y según la versión mostrarlos o enviarlos traducidos.
Atributos Propuestos
ros: Datos de conexión con ROS. **mensajeAnterior**: El último mensaje recibido. **estaEsperando**: Evita que se abran nuevos hilos. **listener**: Datos de escucha con ROS. **traduccion:** Guarda la traducción.
Métodos Propuestos
habla: Establece a dónde se envían los mensajes y los publica. **escuchar**: Inicia un hilo de escucha. **invocaEscuchar**: Cada segundo intenta hacer una escucha. **traducirEsEn**: Traduce el mensaje del castellano al inglés. **traducirEnEs**: Traduce el mensaje del inglés al castellano. **pulsaIntro:** Invoca el método habla cuando se pulsa Intro.

5.5 Análisis de Casos de Uso y Escenarios

5.5.1 Hacer hablar al robot de forma remota

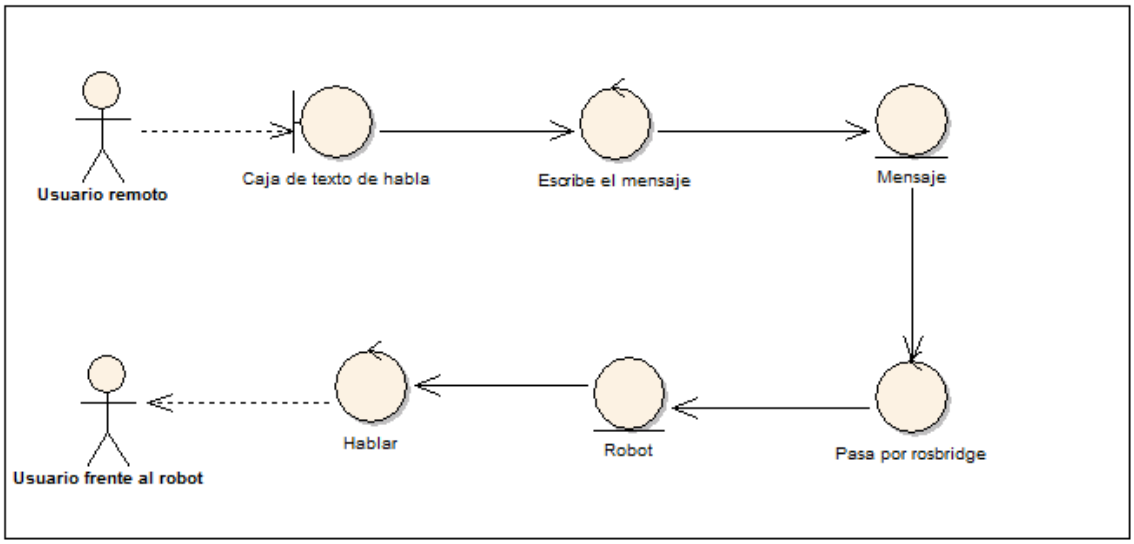


Figura 5.5.1 Diagrama del habla del robot

Hacer hablar al robot	
Precondiciones	La página web tiene que estar dirigida a la IP del robot (puede ser una conexión local) y al puerto del rosbridge. Roscore, rosbridge, sound_play y el programa de habla en el idioma que deseemos deben

	estar activados en el robot. También es necesario tener instalada la voz sintética que se va a utilizar. En caso de que no esté instalada, es necesario instalarla, pero solo es necesario hacerlo una única vez.
Poscondiciones	El nodo de habla del robot recogerá el mensaje y comenzará a hablar por voz hacia el usuario que tenga en frente.
Actores	Usuario remoto: A través de la página web envía mensajes al robot. Robot: Recibe los mensajes y habla. Usuario frente al robot: Es a quien se dirige el mensaje.
Descripción	Permite a un usuario remoto comunicarse con la persona que está frente al robot, lo cual permite dirigirse con voz a una persona que no pueda hablar hacia otra persona, por ejemplo.
Variaciones (escenarios secundarios)	Se puede incluir la variación del idioma, haciendo que los mensajes que se envían al robot sean traducidos a otro idioma.
Excepciones	En el caso del escenario secundario se puede dar el caso de que el servicio externo de traducción no esté disponible en el momento del uso, ya que no tenemos control sobre él.
Notas	

5.5.1.1 Traducción del mensaje

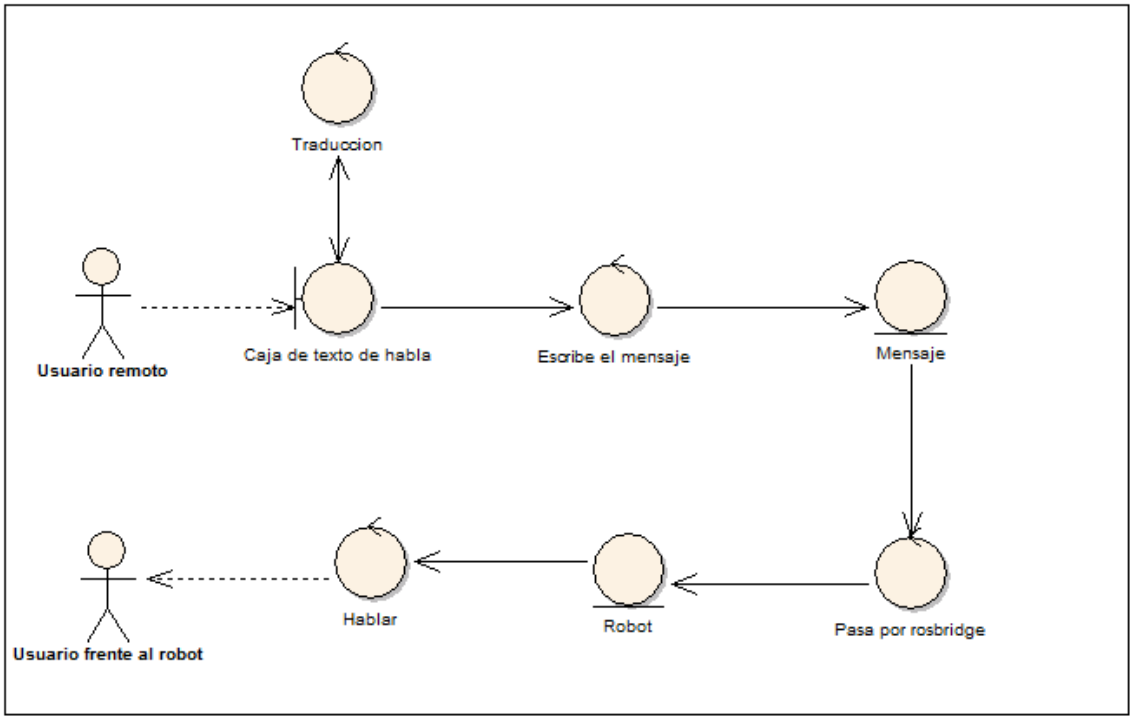


Figura 5.5.1.1 Diagrama del habla del robot con traducción del mensaje

5.5.2 Recibir la escucha del robot

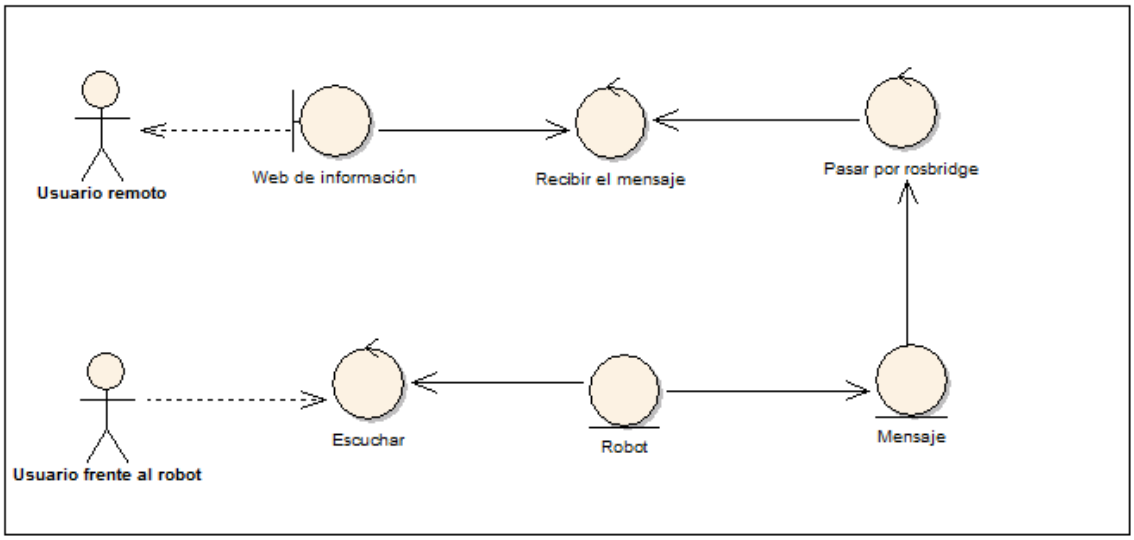


Figura 5.5.2 Diagrama del reconocimiento de voz del robot

Recibir la escucha del robot	
Precondiciones	La página web tiene que estar dirigida a la IP del robot (puede ser una conexión local) y al puerto del rosbridge. Roscore, rosbridge, y pocketsphinx, utilizando el diccionario que especifiquemos, deben estar activados en el robot.
Poscondiciones	El robot permanecerá a la escucha de cualquier sonido y hará una interpretación de lo escuchado para generar mensajes que serán recogidos y mostrados por la web.
Actores	Usuario remoto: A través de la página web recibe mensajes del robot. Robot: Escucha y envía los mensajes. Usuario frente al robot: Es quien habla al robot.
Descripción	Una persona frente al robot o con el micrófono del mismo habla, el robot reconocerá lo que ha dicho para publicarlo como un mensaje que será mostrado en la página web.
Variaciones (escenarios secundarios)	El mensaje enviado puede ser traducido usando un servicio externo antes de ser mostrado al usuario remoto.
Excepciones	En el caso del escenario secundario se puede dar el caso de que el servicio externo de traducción no esté disponible en el momento del uso, ya que no tenemos control sobre él.
Notas	

5.5.2.1 Traducción del mensaje

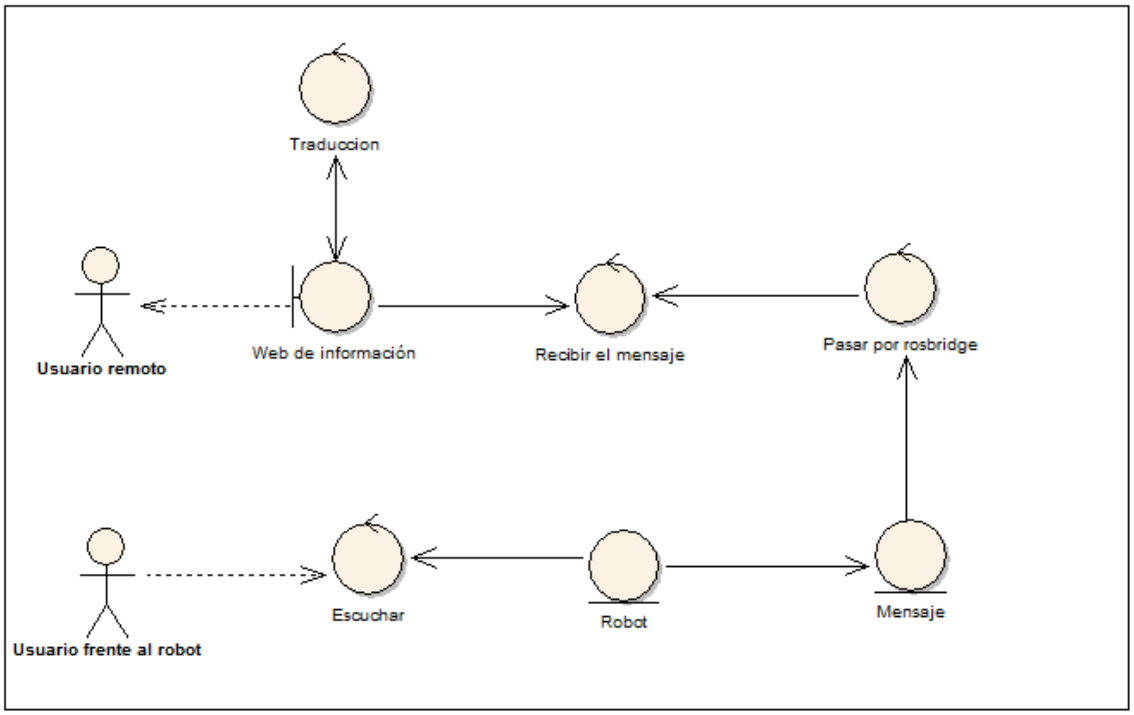


Figura 5.5.2.1 Diagrama del reconocimiento de voz del robot con traducción

5.5.3 Control de la mano robótica de forma remota

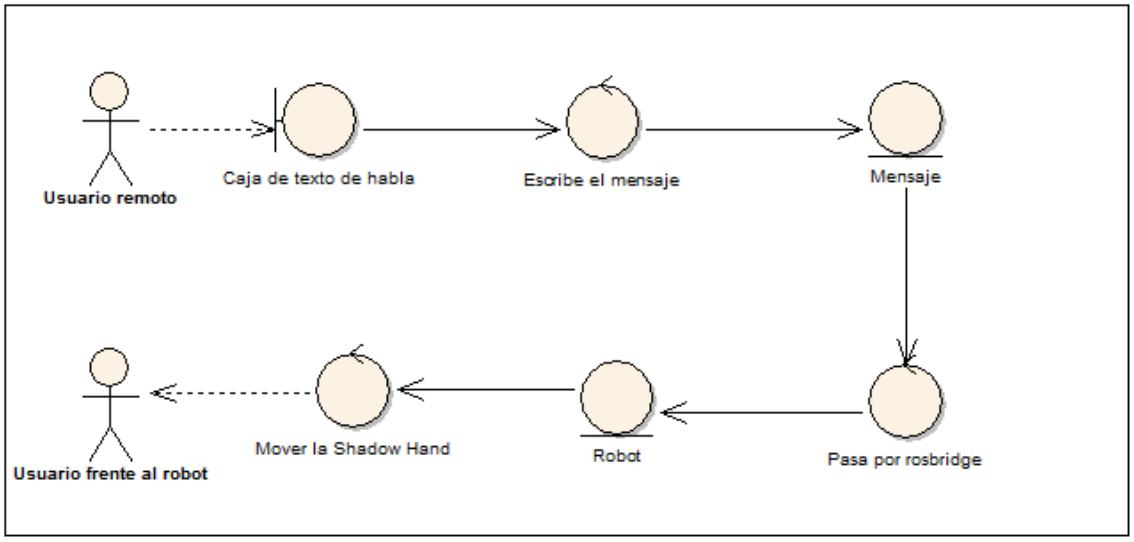


Figura 5.5.3 Diagrama de control de la mano robótica

Control de la mano robótica de forma remota	
Precondiciones	La página web tiene que estar dirigida a la IP del robot (puede ser una conexión local) y al puerto del rosbridge. Roscore, rosbridge y el programa de movimiento de la mano robótica deben estar activados en el robot. En caso de no disponer de la mano real se puede usar una simulación de la misma, con lo cual debe de estar activo el programa de la simulación.

Poscondiciones	El nodo de movimiento de la mano robótica del robot recogerá el mensaje y comenzará a mover la mano para representar los signos.
Actores	Usuario remoto: A través de la página web envía mensajes al robot. Robot: Recibe los mensajes y mueve la mano para representar los signos. Usuario frente al robot: Persona que entienda el lenguaje de signos a la que van dirigidas las señas hechas con la mano.
Descripción	Permite a un usuario remoto comunicarse con la persona que está frente al robot mediante lenguaje de signos.
Variaciones (escenarios secundarios)	Se puede incluir la variación del idioma, haciendo que los mensajes que se envían al robot sean traducidos a otro idioma.
Excepciones	En el caso del escenario secundario se puede dar el caso de que el servicio externo de traducción no esté disponible en el momento del uso, ya que no tenemos control sobre él.
Notas	

5.5.3.1 Traducción del mensaje

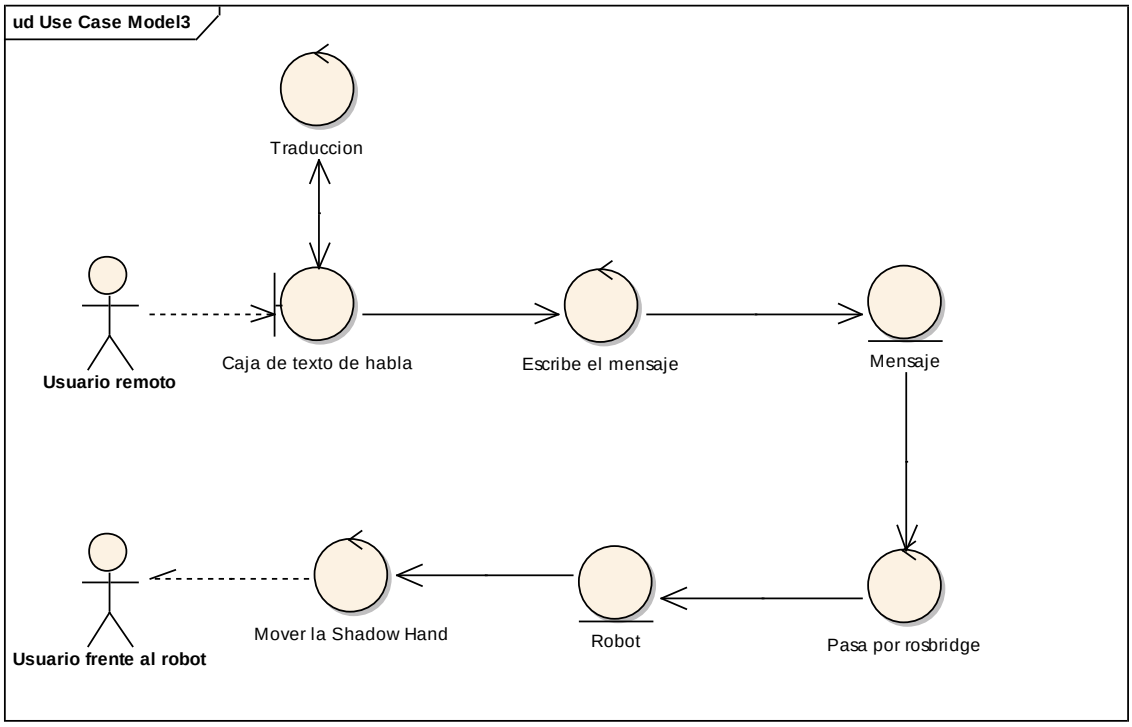


Figura 5.5.3.1 Diagrama de control de la mano robótica con traducción de los mensajes

5.5.4 Control de la mano robótica por voz

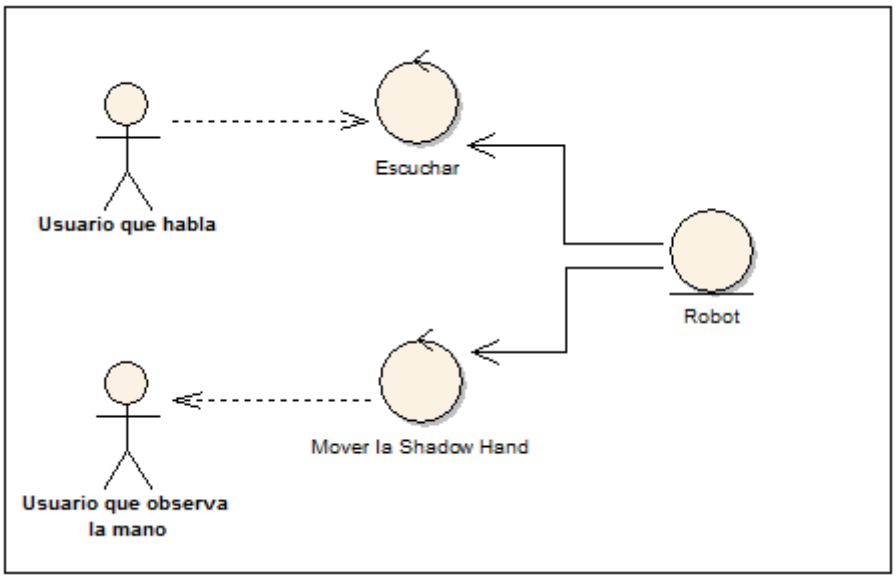


Figura 5.5.4 Diagrama de control de la mano robótica por voz

Control de la mano robótica por voz	
Precondiciones	Roscore, rosbridge, pocketsphinx, utilizando el diccionario que especifiquemos; y el nodo de control de la mano deben estar activados en el robot.
Poscondiciones	El robot permanecerá a la escucha de cualquier sonido y hará una interpretación de lo escuchado para generar mensajes que serán recogidos por el nodo de movimiento de la mano robótica y convertidos en signos por ésta. En caso de no disponer de la mano real se puede usar una simulación de la misma, con lo cual debe de estar activo el programa de la simulación.
Actores	Usuario que habla: Habla al robot. Robot: Escucha y hace una interpretación en lengua de signos con la mano robótica. Usuario que observa la mano: Recibe el mensaje por lengua de signos.
Descripción	Una persona frente al robot o con el micrófono del mismo habla, el robot reconocerá lo que ha dicho para convertirlo en signos con la mano robótica.
Variaciones (escenarios secundarios)	
Excepciones	
Notas	

5.6 Análisis de Interfaces de Usuario

5.6.1 Descripción de la Interfaz

En este proyecto encontramos varios tipos de interfaces, por una parte tenemos las interfaces para comunicarse con el robot de forma remota por medio de un servicio web y luego están las interfaces con el propio robot, las cuales pueden variar en apariencia excepto en lo referido a la mano robótica que se utiliza.

5.6.1.1 Interfaz web

Esta interfaz nos permite establecer una comunicación remota con el robot, es una página web sencilla en la que se muestran los mensajes que se reciben en la parte superior (con deslizamiento automático si se llenase) y se permiten enviar nuevos mensajes con la caja de texto y el botón de la parte inferior. Más adelante se detallará su funcionamiento.

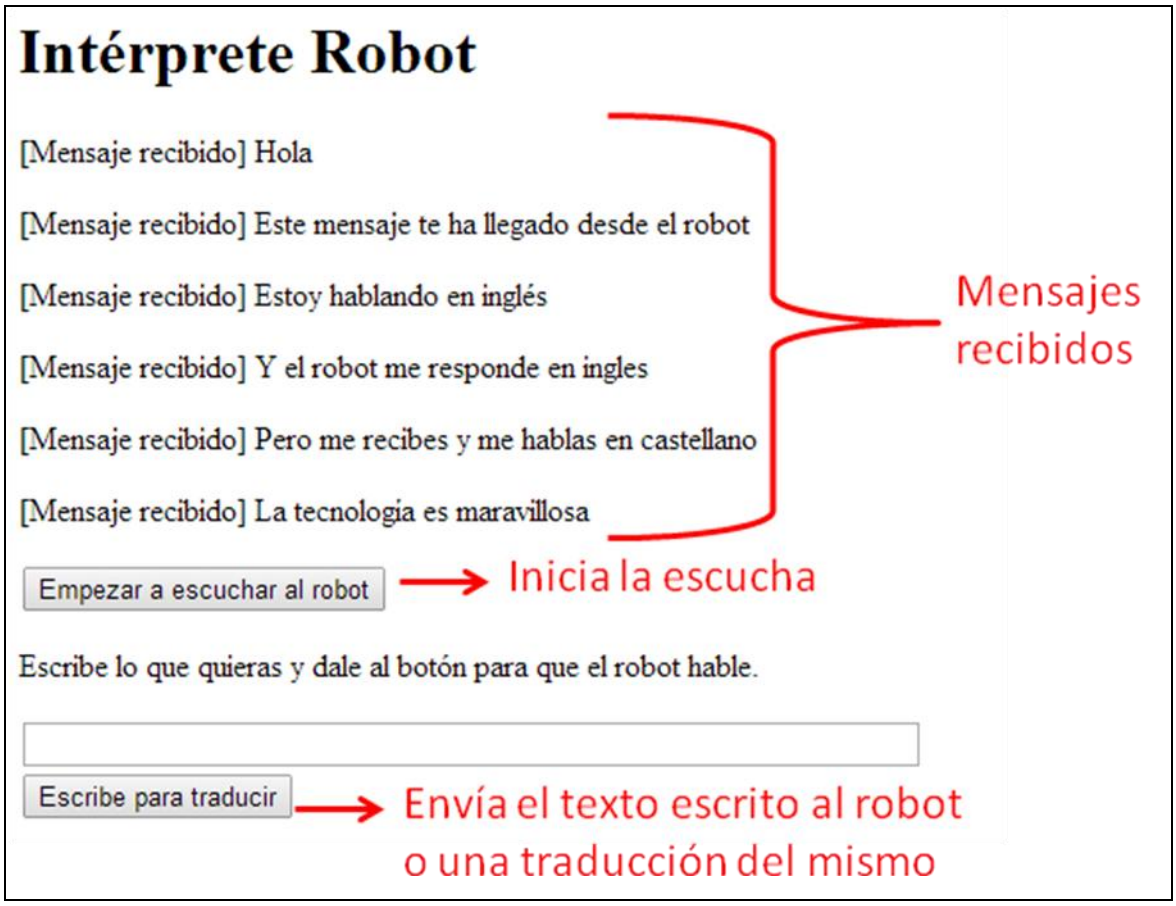


Figura 5.6.1.1 Boceto de una interfaz

5.6.1.2 Interfaz de escucha

Se trata de un micrófono que puede estar en el propio robot o conectado a él. Si el robot dispone de pantalla se puede ver cómo funciona el reconocimiento de voz, qué vocablos reconoce y qué palabras interpreta.

El programa de escucha se encargará de reconocer lo que el usuario dice para convertirlo en texto.

5.6.1.3 Interfaz de habla

A través de los altavoces del robot o conectados al mismo se escucha por voz sintetizada los mensajes que el robot recibe. Existen dos versiones, una para castellano y otra para inglés.

5.6.1.4 Mano robótica

La mano robótica o la simulación de la misma es otra interfaz, ya que nos permite de forma visual comunicarnos con otra persona mediante lenguaje de signos.

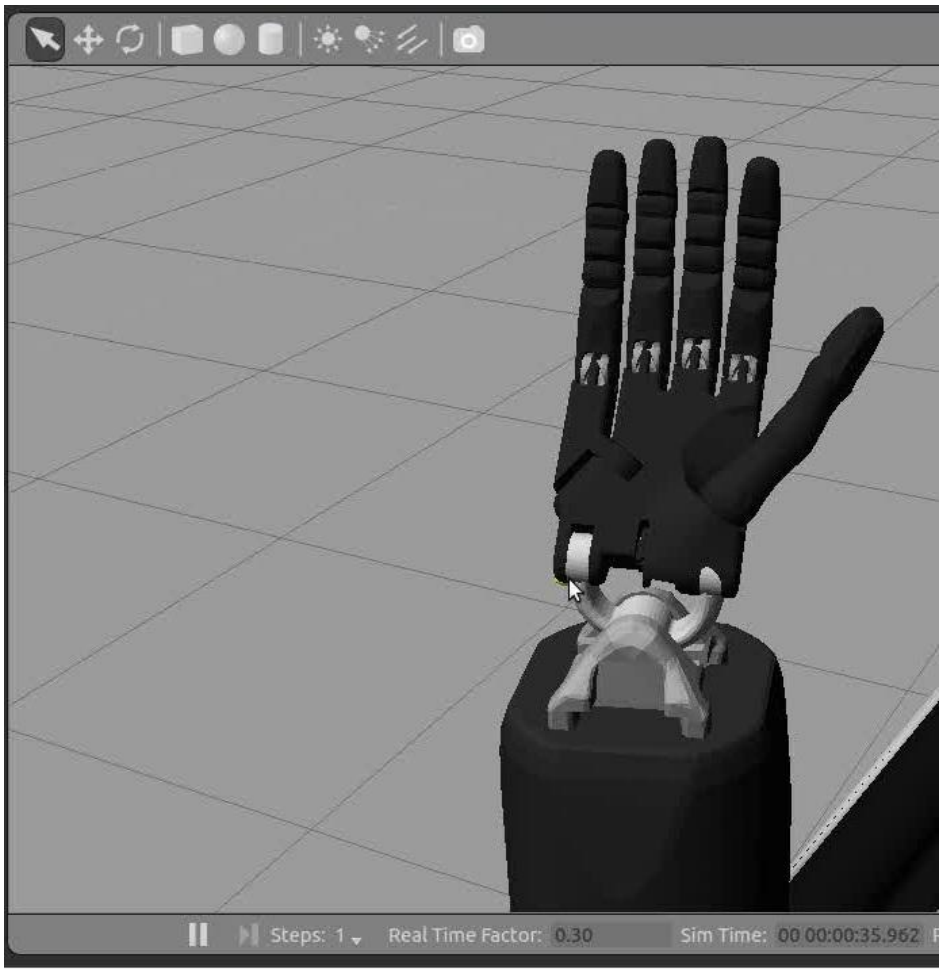


Figura 5.6.1.4 Simulación de la Shadow Hand en Gazeboo

5.6.2 Descripción del Comportamiento de la Interfaz

5.6.2.1 Interfaz web

Esta interfaz nos permite establecer una comunicación remota con el robot ya que establece una conexión con el nodo rosbridge, si este está correctamente dirigido a la IP del robot en el código javascript interno.

En las pruebas se han hecho conexiones locales, ya que la IP en las máquinas usadas no es fija y cambia con el reinicio del router, pero en un contexto real se pretende que las IP fueran fijas y la página estuviera publicada en un servidor web con la IP fija ya escrita dentro de su código, ya que no se pretende complicar al usuario con cajas de texto extra para especificar la IP y el puerto en el que está el robot.

Una vez abierta la interfaz en un navegador web hay que pulsar el botón "Empezar a escuchar al robot" para que se establezca la conexión de escucha, de este modo empezarán a aparecer todos los mensajes que se reciban del robot, y cuando estos llenen la pantalla entonces se producirá deslizamiento hacia abajo de forma automática, siempre dejando a la vista la caja de texto y el botón para escribir nuevos mensajes, como ocurre en cualquier servicio de chat.

Para enviar mensajes basta escribirlos en la caja de texto y pulsar el botón para enviar. Según la versión de la página web, además de enviar y recibir mensajes, estos son traducidos de forma automática por un servicio externo y sin falta de interacción por parte del usuario.

5.6.2.2 Interfaz de escucha

Para la escucha se utiliza Pocketsphinx[16-17], que es un nodo de ROS que a partir de un diccionario predefinido hace un reconocimiento de la pronunciación y crea una interpretación de lo que ha escuchado que, a su vez, publica en un canal de escucha, que es el que utilizan otros nodos según nos interese mostrarlo en la web o interpretarlo con la mano.

Esta aplicación nos muestra en una ventana de línea de comandos los vocablos y las palabras que va reconociendo, así como su interpretación final, que es lo que publica.

5.6.2.3 Interfaz de habla

Mediante el uso de un nodo el texto recibido para su habla es convertido en voz sintética. Este nodo tiene dos versiones, una para recibir texto en inglés y generar una voz en inglés y otro para recibir texto en castellano y generar una voz en castellano.

Se pueden instalar nuevas voces en el equipo y crear variantes en otros idiomas o con otras entonaciones de voz.

5.6.2.4 Mano robótica

Mediante un nodo, que puede recibir los datos de la interfaz de escucha o de lainterfaz web, se transforma la cadena de texto recibida en órdenes de movimiento de la mano. La mano va tomando la posición de cada letra para luego restaurarse a una posición neutral (mano abierta, dedos rectos) antes de realizar el siguiente signo. Si no se da tiempo a que se realice esta restauración, los dedos podrían tropezar y producir un error en la simulación.

La mano puede ser la versión real de la Shadow Hand o una simulación en Gazeboo de la misma, lo cual es más accesible para todo el público.

5.6.3 Diagrama de Navegabilidad

El diagrama de navegabilidad se puede dividir en dos partes: Una para el envío de mensajes y otra para la recepción de los mismos. Al enviar un mensaje, el comportamiento del robot cambiará, pero para recibirlos es necesario primero iniciar el proceso de escucha si se pretende recibir información del robot, lo cual modifica la pantalla cada vez que se recibe un mensaje. Se ha incluido en este diagrama el comportamiento del robot ya que también es parte de la interfaz.

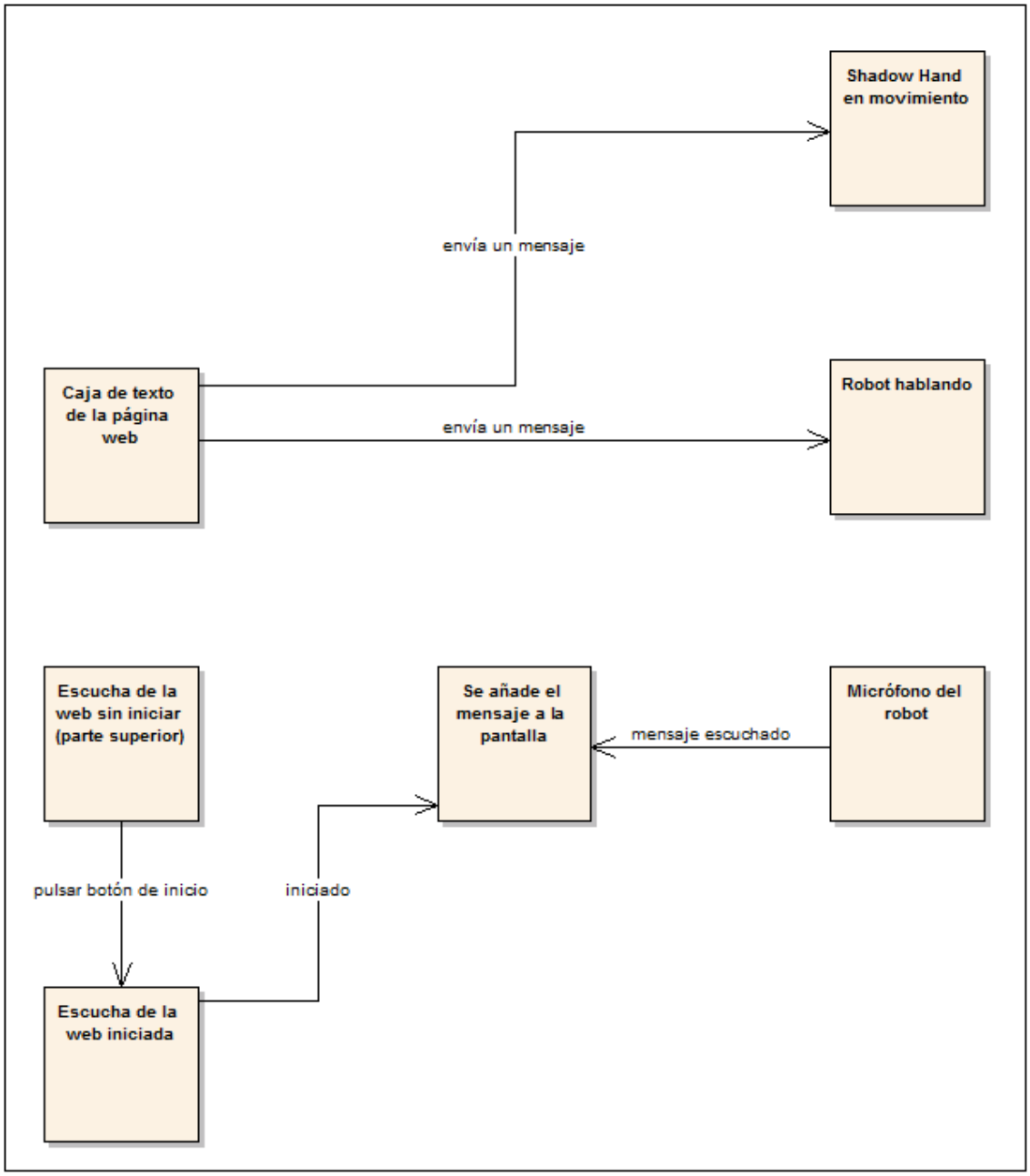


Figura 5.6.3 Diagrama de Navegabilidad

5.7 Especificación del Plan de Pruebas

Aquí, además de exponer las pruebas de los casos de uso, también se incluyen las pruebas de los componentes fuera de estos durante su desarrollo antes de completar su funcionamiento.

5.7.1 Pruebas unitarias

Dado que el proyecto consiste en varias partes de un sistema funcionando en colaboración entre las mismas, las pruebas unitarias se limitan a comprobar el funcionamiento de los componentes con datos predefinidos, de modo que no necesitan ni recibir datos de entrada ni publicar datos en un canal de escucha.

Caso de Uso 1: Síntesis de voz	
Prueba	**Resultado Esperado**
Sintetizar voz	El sistema sintetiza voz con un texto predefinido en inglés con la voz predeterminada.
Prueba	**Resultado Esperado**
Sintetizar voz en castellano	El sistema sintetiza voz con un texto predefinido en castellano con la otra voz instalada.

Caso de Uso 2: Movimiento de la Shadow Hand (mano robótica)	
Prueba	**Resultado Esperado**
Mover las articulaciones de los dedos.	Las articulaciones de la mano se mueven según las órdenes recibidas.
Prueba	**Resultado Esperado**
Ordenar mover las articulaciones de los dedos fuera de sus límites.	Las articulaciones llegan a su posición máxima o mínima sin sobrepasarla.
Prueba	**Resultado Esperado**
Ejecutar un programa para cerrar el puño (pulgar fuera)	Los dedos se mueven simultáneamente para tomar la posición de puño cerrado, esperando el pulgar un tiempo para que el pulgar se coloque después de moverse las demás articulaciones.
Prueba	**Resultado Esperado**
Ejecutar un programa para abrir la mano (pulgar dentro)	Los dedos se mueven simultáneamente para tomar la posición de mano abierta, siendo el pulgar el primer dedo en moverse antes de que se inicie la acción de apertura de los demás dedos.
Prueba	**Resultado Esperado**
Ejecutar un programa para que la mano haga un signo	Los dedos se mueven simultáneamente para que la mano tome la forma de un signo del alfabeto dactilológico.
Prueba	**Resultado Esperado**
Ejecutar un programa para que la mano realice una secuencia de signos.	Los dedos se mueven simultáneamente para que la mano tome la forma de una secuencia de signos del alfabeto dactilológico. Tras cada signo la mano tendrá un tiempo para volver a una posición neutral antes de cambiar al siguiente signo.

Caso de Uso 3: Reconocimiento de voz (Pocketsphinx)	
Prueba	**Resultado Esperado**
Reconocer frase	Pocketsphinx reconoce las frases del diccionario utilizado.

5.7.2 Pruebas de Integración

Caso de Uso 3: Enviar y recibir mensajes entre el robot y la página web	
Prueba	**Resultado Esperado**
Enviar mensaje al robot desde la página web	El robot recibe el mensaje y este se publica.
Prueba	**Resultado Esperado**
Iniciar escucha en el mismo canal de publicación	Cada vez que se envía un mensaje, este se publica en un canal indicado, y al ser el mismo canal que el de escucha para estas pruebas, el mensaje recibido en el robot es leído por la página web y publicado en la misma.
Prueba	**Resultado Esperado**
Enviar mensajes traducidos	Manteniendo el mismo canal de escucha para estas pruebas, los mensajes enviados son traducidos antes de ser enviados y al recibirse se reciben traducidos.
Prueba	**Resultado Esperado**
Traducir los mensajes recibidos	Manteniendo el mismo canal de escucha para estas pruebas, los mensajes recibidos son traducidos al recibirse y aparecen traducidos.

5.7.3 Pruebas del sistema

Caso de Uso 4: Controlar el robot con el servicio web	
Prueba	**Resultado Esperado**
Hacer hablar al robot desde la página web.	El robot recibirá el mensaje y dirá el mensaje.
Prueba	**Resultado Esperado**
Hacer mover la Shadow Hand al robot desde la página web.	El robot recibirá el mensaje y moverá la Shadow Hand (o su simulación) para representar el mensaje recibido.
Prueba	**Resultado Esperado**
Recibir mensaje escuchado del robot.	En la página web se publican los mensajes escuchados por el robot.
Prueba	**Resultado Esperado**
Hacer hablar al robot desde la página web un mensaje traducido.	El robot recibirá un mensaje traducido de la web y dirá el mensaje traducido.
Prueba	**Resultado Esperado**
Hacer mover la Shadow Hand al robot desde la página web con un mensaje traducido.	El robot recibirá el mensaje y moverá la Shadow Hand (o su simulación) para representar el mensaje recibido, que será una traducción del mensaje escrito en la web.
Prueba	**Resultado Esperado**
Recibir mensaje escuchado del robot y mostrarlo traducido.	En la página web se publican los mensajes escuchados por el robot previamente traducidos.

Caso de Uso 5: Controlar la Shadow Hand por voz	
Prueba	**Resultado Esperado**
Traducir voz a lenguaje de signos.	El robot escuchará y convertirá el mensaje interpretado en lenguaje de signos moviendo la Shadow Hand.

Capítulo 6. Diseño del Sistema

6.1 Arquitectura del Sistema

6.1.1 Diagramas de Paquetes

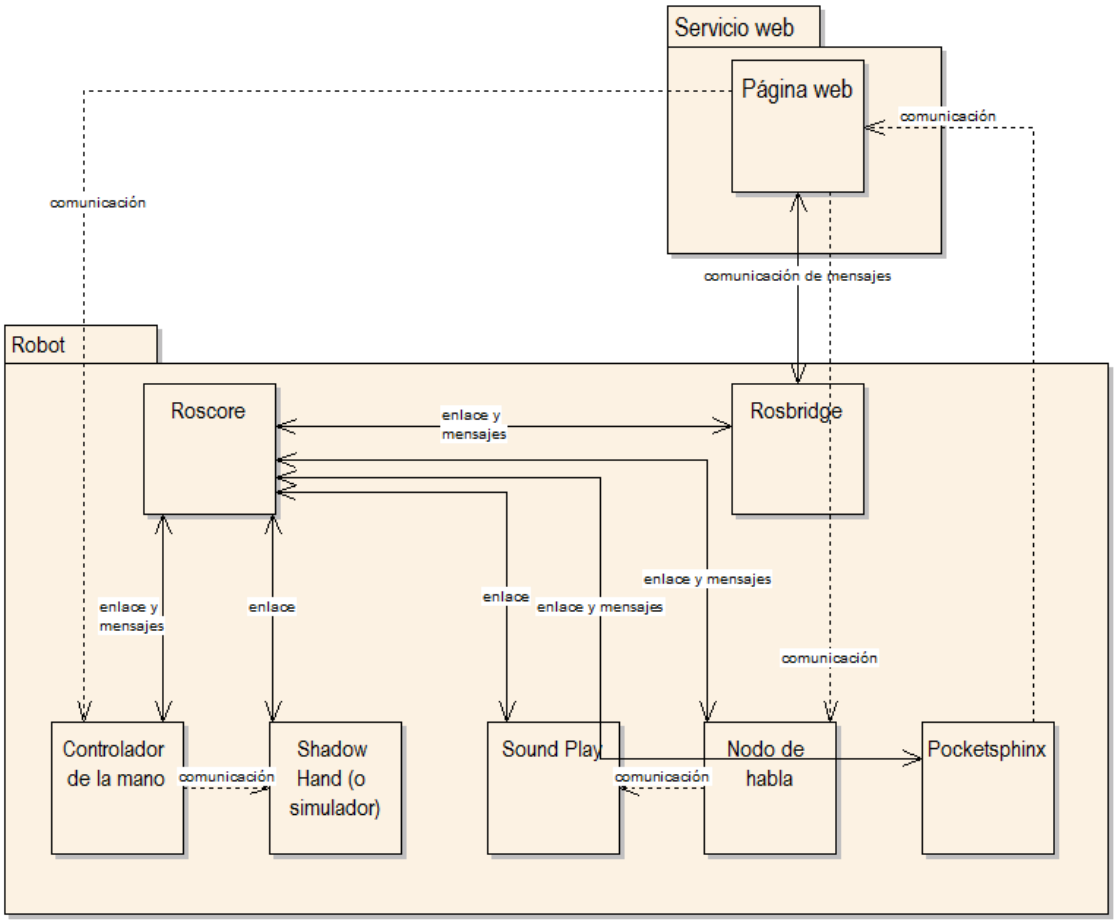


Figura 6.1.1 Diagrama de Paquetes

6.1.1.1 Robot

Contiene los nodos que funcionan en el robot, cada uno con su función, y que se comunican entre sí. Las comunicaciones pasan por roscore, si las flechas tienen una línea de puntos significa que hay una comunicación entre ambos nodos, es decir, hay una relación de dependencia, pero esta no es de forma directa.

- **Roscore**: Núcleo que establece las comunicaciones entre todos los nodos por medio de canales.

- **Rosbridge**: Permite comunicarse con otros programas e incluso desde fuera de la máquina mediante una conexión de red. Permite que los nodos del robot y el servicio web se comuniquen.
- **Controlador de la mano**: Recibe mensajes y los transforma en órdenes de movimiento para la mano.
- **Shadow Hand (o simulador)**: Mano robótica o simulación de la misma, que obedece a las órdenes del controlador.
- **Sound_Play**: Genera audio, en este caso síntesis de voz.
- **Nodo de habla**: Recibe un mensaje y envía la orden de generar una voz sintética determianda.
- **Pocketsphinx**: Es el nodo de reconocimiento de voz. Los mensajes interpretados son publicados en un canal determinado el cual la página web podrá leerlo a través Rosbridge.

6.1.1.2 Servicio web

Contiene únicamente la **página web** que se comunica con el robot. Envía órdenes al controlador de la mano robótica y al nodo de habla, pero de forma indirecta, de ahí que se represente la relación con puntos; ya que a través de Rosbridge publica en los canales que pasan por el núcleo, Roscore, y de estos canales son de los que escuchan dichos nodos para hablar o mover la mano.

No se incluye el servicio de traducción ya que es un servicio externo al proyecto.

6.1.2 Diagramas de Componentes

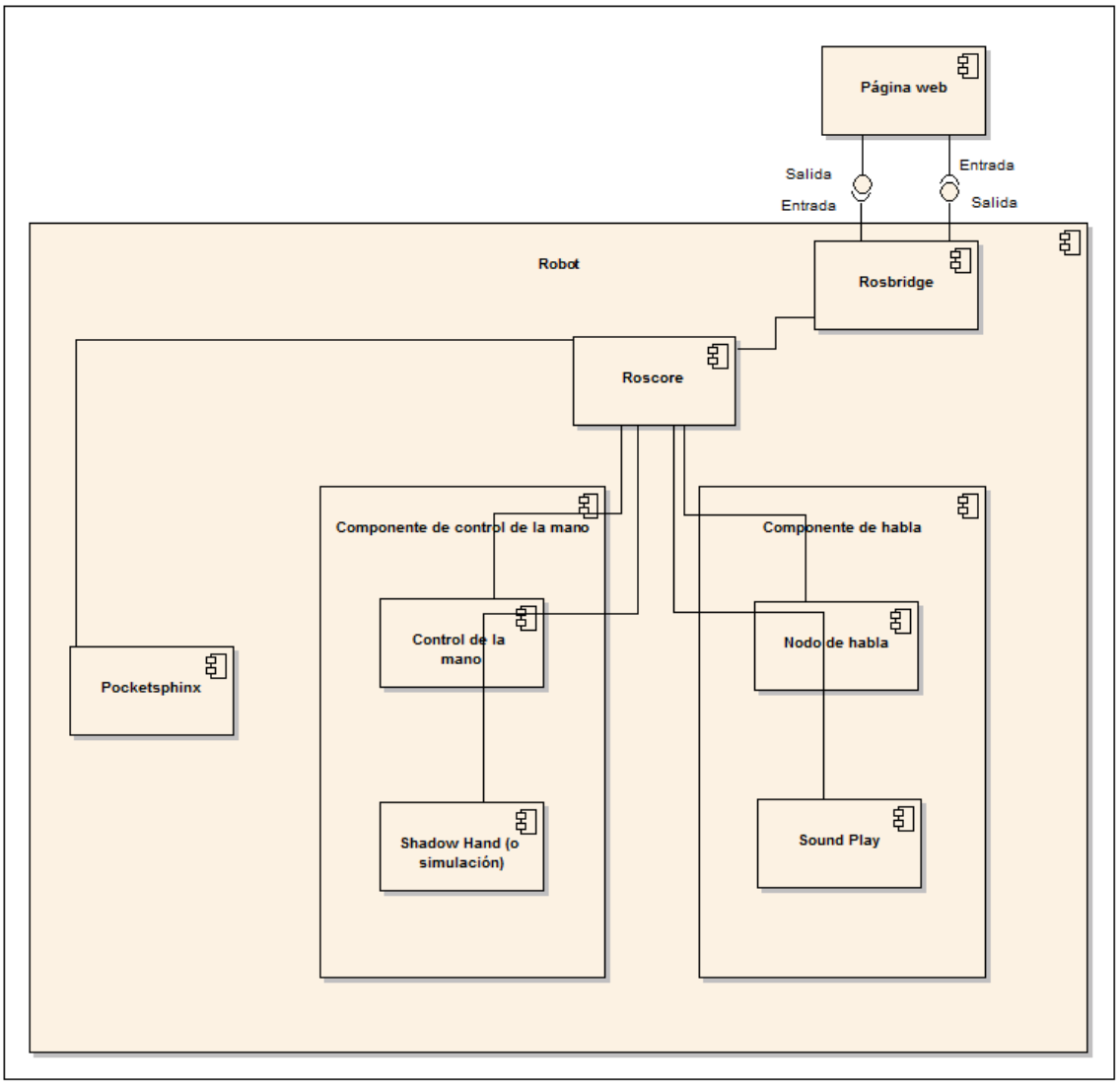


Figura 6.1.2 Diagrama de Componentes

La descripción de los componentes menores va acorde con la descripción de los paquetes. Se han unido el nodo de Control de la mano y Shadow Hand (o simulación) en un mismo **Componente de control de la mano**, y para el nodo de habla y el Sound Play, ambos se han unido en un mismo **Componente de habla**.

En el diagrama se puede ver la relación de entrada y salida que hay entre la página web, que envía y recibe, con el Rosbridge, que también envía y recibe para mantener el contacto del robot con la página web.

6.1.3 Diagramas de Despliegue

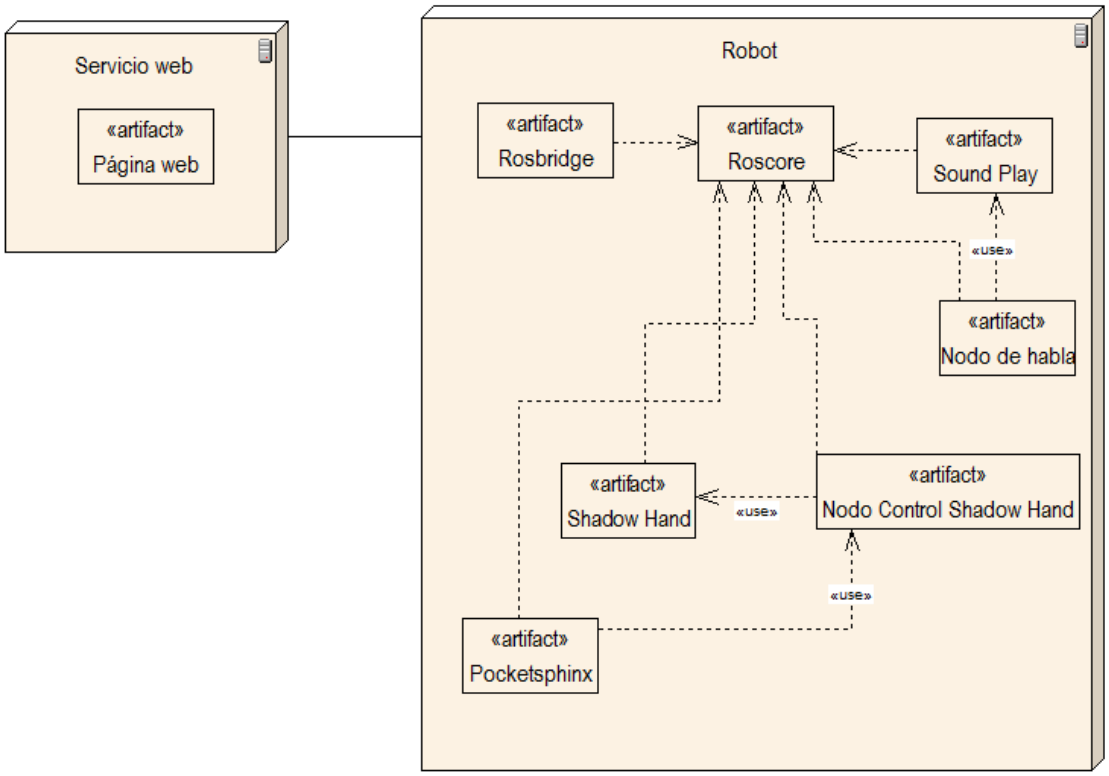


Figura 6.1.3 Arquitectura del sistema

6.1.3.1 Servicio web

Compuesto únicamente con una página web dirigida a la IP del robot para comunicarse con sus nodos, publicando y leyendo mensajes, gracias a Rosbridge.

6.1.3.2 Robot

Formado por sus nodos, comunicados entre sí a través del nodo Roscore.

- **Roscore** es el nodo donde se publican los mensajes en distintos canales a los que se suscriben los demás nodos, como si fuera un tablón de noticias.
- **Rosbridge** es el nodo que permite la comunicación con aplicaciones externas, incluso si no son nodos de ROS. En este caso con la página web, permitiéndole suscribirse y publicar en canales de Roscore.
- **Sound Play** es el nodo que se encarga de reproducir sonidos y de usar el sintetizador de voz.
- El **nodo de habla** es el encargado de recoger los mensajes y de hacer que Sound Play reproduzca con el sintetizador de voz el texto con la voz deseada.

- **Pocketsphinx** se encarga del reconocimiento de voz. El texto reconocido se publica en Roscore para que sea leído por la página web a través de Rosbridge o para que sea interpretado por la Shadow Hand mediante su nodo de control.

- El **nodo de control de la Shadow Hand** a partir del texto enviado por la página web o a partir de lo reconocido por Pocketsphihnx enviará las órdenes a la Shadow Hand para cambiar su posición y hacer los getos.

- La **Shadow Hand**, o su simulación en **Gazebo**, realiza el lenguaje de signos dactilológico a partir de las órdenes enviadas por el nodo de control, que le envía la posición en radianes de sus articulaciones.

6.2 Diagramas de Interacción y Estados

6.2.1 Hacer hablar al robot de forma remota

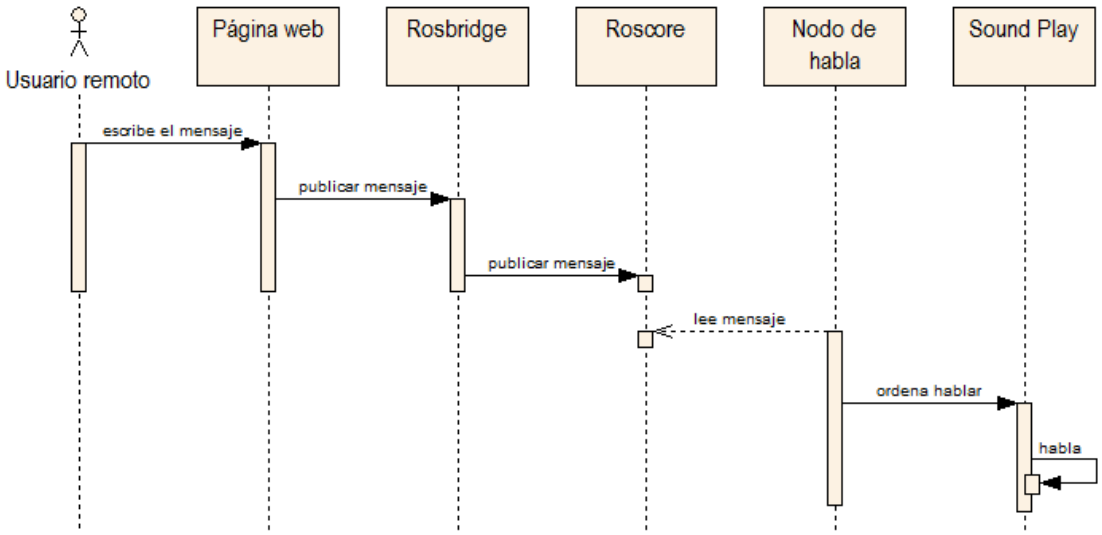


Figura 6.2.1 Diagrama de Interacción sobre el control remoto del habla del robot

El usuario remoto escribe un mensaje en la página web que se envía al nodo Roscore del robot a través del nodo Rosbridge del mismo, publicándose el mensaje en el canal indicado del Roscore, al que permanece a la escucha el Nodo de habla, que lee el mensaje para enviar las órdenes de síntesis de voz a Sound Play.

6.2.2 Recibir la escucha del robot

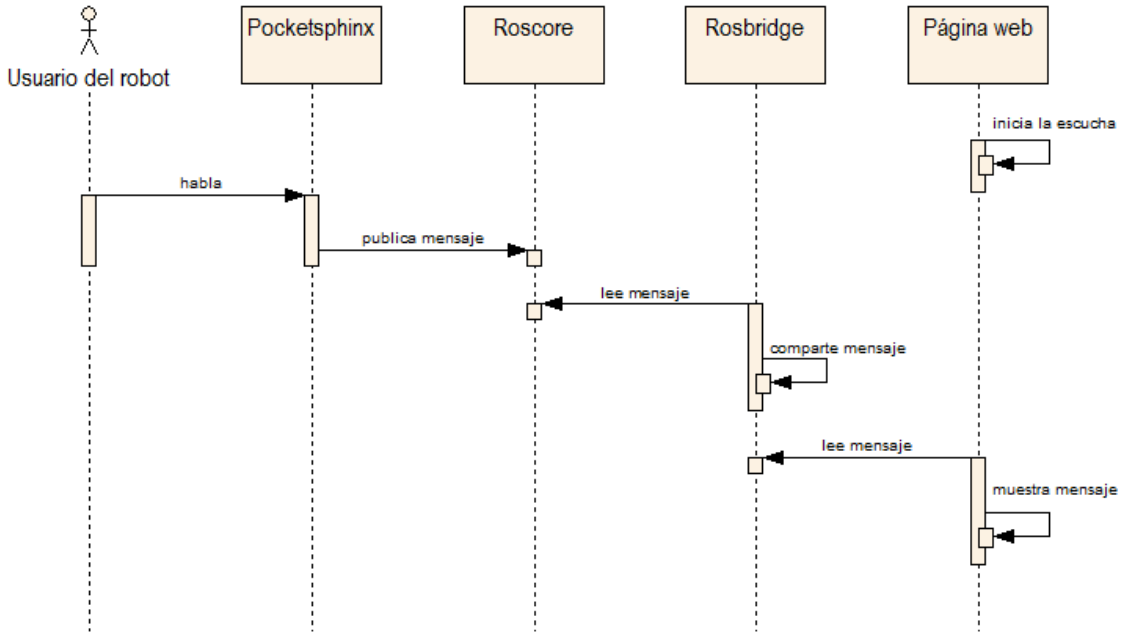


Figura 6.2.2 Diagrama de Interacción sobre la escucha del robot

El robot escucha lo que dice el usuario gracias a Pocketsphinx, que convierte lo dicho en texto y lo publica en Roscore. Rosbridge lee el mensaje de Roscore y lo comparte, de modo que la Página web, si ha activado la escucha, leerá el mensaje a través de Rosbridge y lo mostrará.

6.2.3 Control de la mano robótica de forma remota

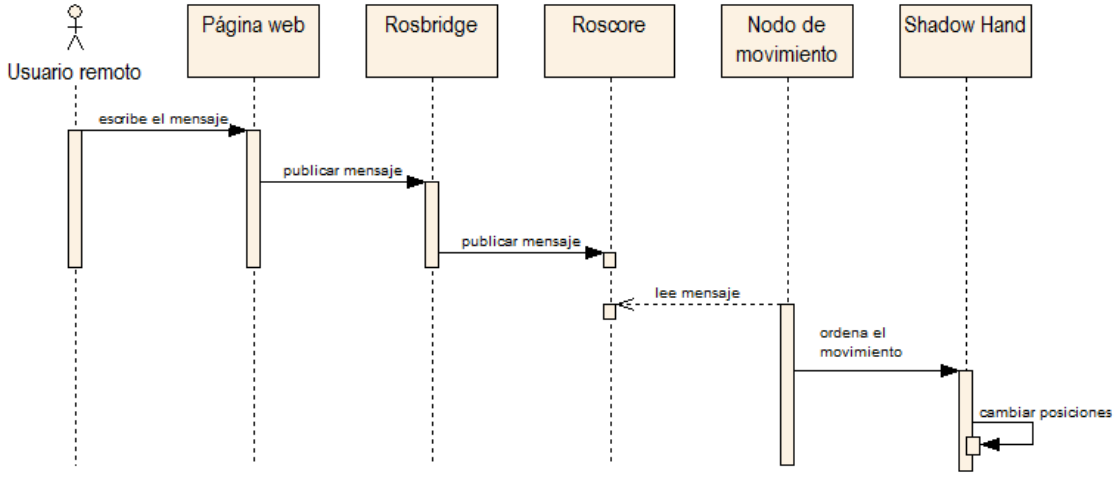


Figura 6.2.3 Diagrama de Interacción sobre el control de la mano robótica

Como ocurría en el caso de hacer hablar al robot, a través de la página web el usuario escribe un mensaje, que llega al núcleo del robot a través de Rosbridge y que es leído por el nodo de movimiento de la Shadow Hand para enviar las órdenes de movimiento a la Shadow Hand o su simulación.

6.2.4 Control de la mano robótica por voz

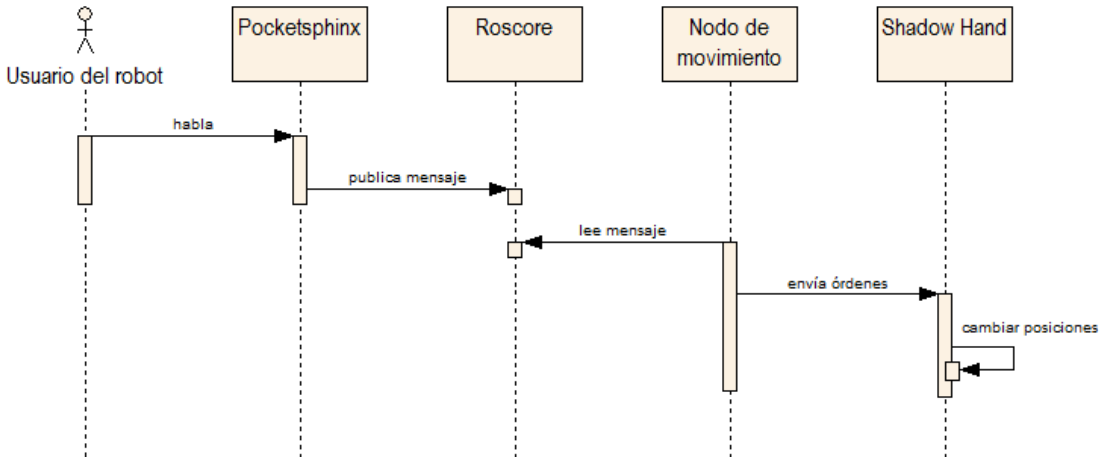


Figura 6.2.4 Diagrama de Interacción sobre el control por voz de la mano robótica

En este caso no es necesario que intervenga Rosbridge, ya que no se necesita una comunicación con la página web. Con Pocketsphinx se convierte lo escuchado en un mensaje que se publica en Roscore, para luego ser leído por el nodo de movimiento de la Shadow Hand, suscrito al canal, para luego enviar las órdenes a la Shadow Hand o su simulación.

6.3 Diagramas de Actividades

Los nodos permanecen siempre suscritas a un canal de publicación para que en caso de se publique un mensaje en ese canal, para recogerlo e interactuar sobre el mensaje recibido. El siguiente gráfico es genérico para todos los nodos programados en el proyecto:

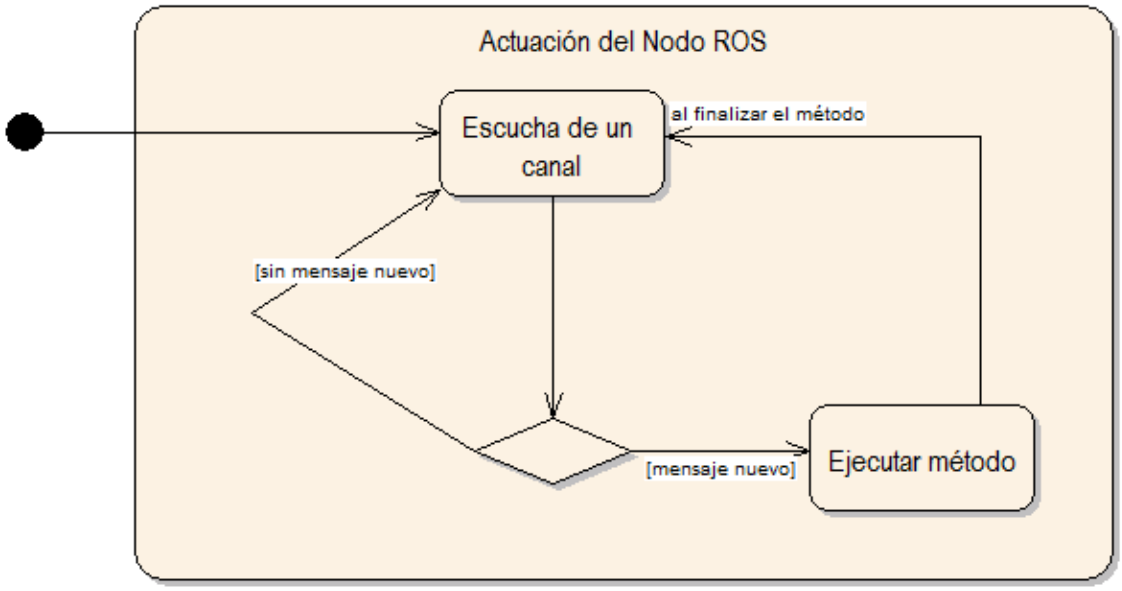


Figura 6.3 Diagrama de actividad genérico de los nodos

El mismo diagrama también es aplicable al método javascript de la página web que permanece a la escucha gracias a Rosbridge, pero no es del todo igual para la página web que mantiene dos procesos simultáneos.

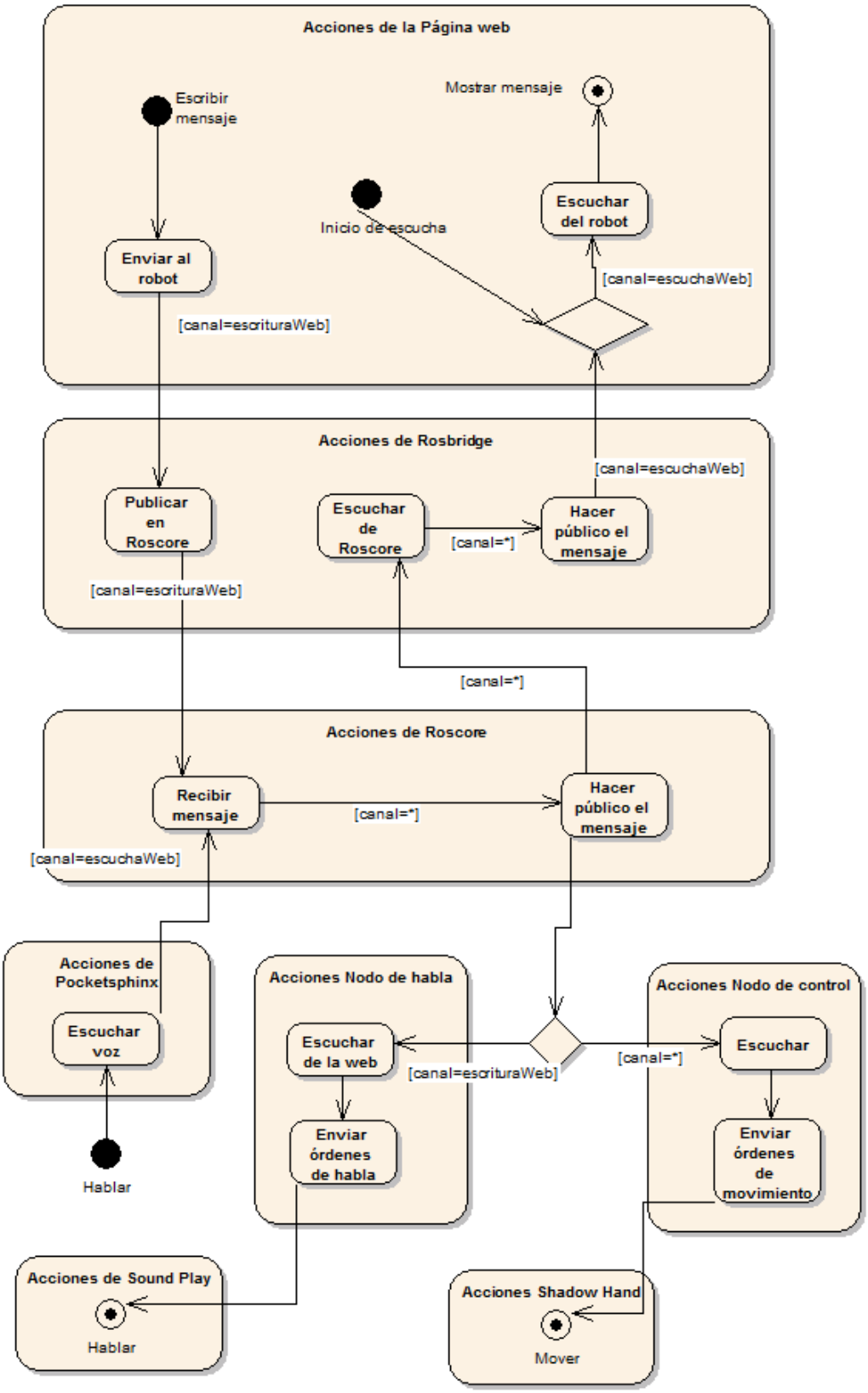


Figura 6.3.1 Diagrama de actividad del sistema completo

Este diagrama viene a representar el sistema completo, como se vió en apartados anteriores. Los mensajes se publican en canales, de modo que para entender bien el diagrama se diferencian los canales de escrituraWeb (lo que se recibe de la web) y el de escuchaWeb (lo que se envía a la web), siendo * referido a ambos canales, siendo el caso del Nodo de control de la Shadow Hand el único que escucha de cualquiera de los dos canales.

6.4 Diseño de la Interfaz

6.4.1 Interfaz web

Es una interfaz sencilla que simula un chat online. En la parte superior aparecen los mensajes, mientras que en la parte inferior se escriben los mensajes. Hay que pulsar un botón para iniciar la escucha del robot, que será continua a partir de ese momento, de modo que es a partir de ahí cuando se empiezan a recibir los mensajes del robot. La página se desliza hacia abajo de forma automática cuando se llena de texto, como ocurriría en un chat online.

Figura 6.4.1 Muestra de la interfaz de la web

6.4.2 Interfaz del robot

El robot puede recibir mensajes de su usuario a través de la voz para comunicarse con el usuario remoto, pero a la vez puede responder con los mensajes que reciba de la web con voz. También puede, tanto para los mensajes que reciba por voz como los que reciba a través de la web, usar la mano robótica para hacer un deletreo dactilológico de los mensajes recibidos.

Para algunas letras los dedos tienen que representar movimiento, como en el caso de la 'V' o la 'Z'.

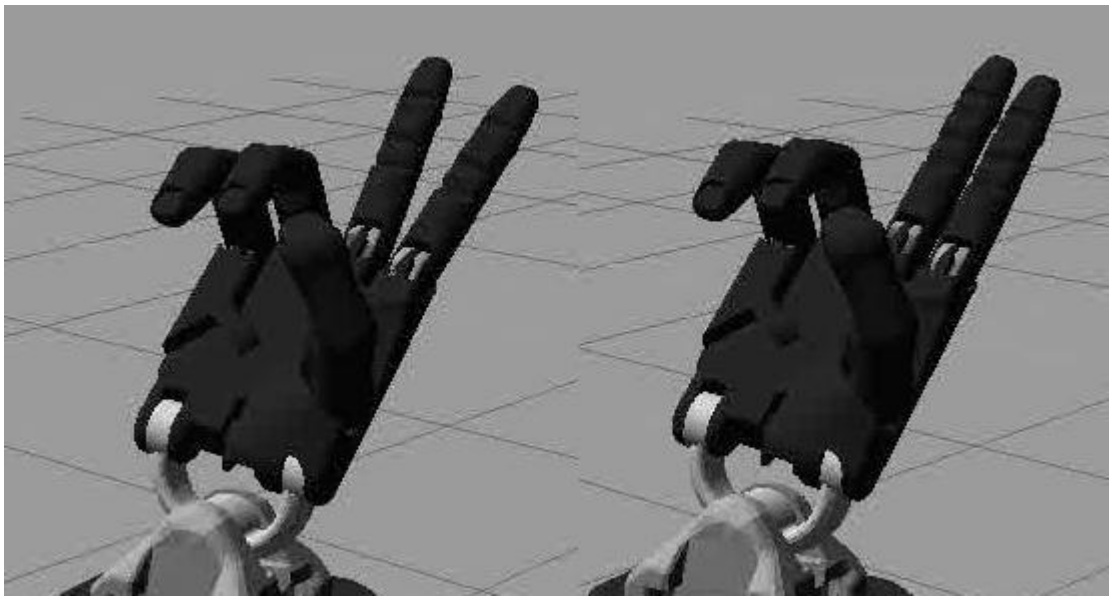


Figura 6.4.2 Posiciones de la Shadow Hand al representar la V

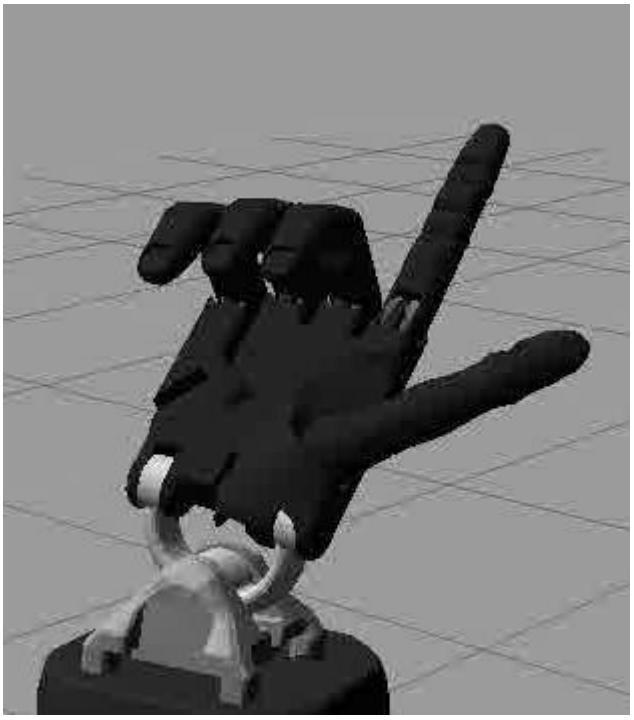


Figura 6.4.2.1 Posiciones de la Shadow Hand al representar la L

6.5 Especificación Técnica del Plan de Pruebas

6.5.1 Pruebas Unitarias

6.5.1.1 Síntesis de voz

6.5.1.1.1 Sintetizar voz

Se crea el nodo de habla con una frase predefinida, se inician Roscore y Sound Play para luego ejecutar el Nodo de habla y que se escuche la voz sintetizada.

Un error común al realizar estas pruebas es ejecutar el nodo después de ejecutar Roscore sin haber activado Sound Play, tras lo cual devuelve un mensaje de error indicando que Sound Play no está activado.

6.5.1.1.2 Sintetizar voz en castellano

Es el mismo caso que el anterior, pero esta vez al nodo de habla se le añade una línea de código que cambia la voz por defecto a una especializada para hablar en castellano que ha sido instalada con anterioridad. El objetivo es que reproduzca un mensaje predefinido en castellano.

6.5.1.1.3 Enviar orden de hablar cuando ya está hablando

Se adapta el nodo de habla para recibir mensajes y se hace una primera prueba para que el programa escuche de un canal y reproduzca el mensaje recibido, este mensaje lo puede recibir desde la página web (para lo cual en este caso también hay que tener Rosbridge activado), coincidiendo con una prueba de integración, y también desde el propio nodo de publicación de prueba del tutorial de ROS si el nodo de habla está suscrito al mismo canal en el que publica este[2].

Al principio si se envía un mensaje sin que Sound Play haya acabado de hablar, interrumpirá el que estaba reproduciendo para reproducir el nuevo. Esto es normal en el funcionamiento.

Para dar tiempo a que se reproduzcan los mensajes cortos, se hace "dormir" al nodo de habla con rospy.sleep para que no vuelva a leer de Roscore otro mensaje en cinco segundos.

El objetivo de esta prueba es que se le envíen constantemente mensajes al nodo de habla interrumpiéndose solo a partir de los cinco segundos.

Los mensajes pueden ser automatizados con el nodo de prueba anteriormente mencionado, ya que envía mensajes en intervalos cortos, o enviándolos por la página web.

[2]Writing the Publisher Node: http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber(c%2B%2B)

6.5.1.2 Movimiento de la Shadow Hand (mano robótica)

6.5.1.2.1 Mover las articulaciones de los dedos

Usando la simulación de la Shadow Hand se prueban las distintas articulaciones indicadas en las especificaciones[3] con distintos valores en radianes. Para ello se tienen activados el Roscore y la simulación de la mano en Gazebo. Un ejemplo de código, que se puede encontrar en el tutorial de la Shadow Hand[4], es el siguiente código que paso a detallar:

```
rostopic pub /sh_ffj3_position_controller/command
std_msgs/Float64 1.5
```

En rojo está el código de la articulación y en azul está la posición que debe adoptar en radianes.

El propósito de esta prueba es probar todas las articulaciones y conocer los límites de estas.

6.5.1.2.2 Ordenar mover las articulaciones de los dedos fuera de sus límites

Prueba similar a la anterior, con el añadido que a partir de las especificaciones se conocen los límites con las especificaciones para saber hasta dónde se pueden mover las articulaciones en caso de sobrepasar estos límites.

El resultado esperado es que las articulaciones frenen al llegar a su límite.

6.5.1.2.3 Ordenar chocar dos dedos

Se produce el choque de dos dedos (como un cruce de dedos) para comprobar que las articulaciones se paran al darse el caso.

6.5.1.2.4 Ejecutar un programa para cerrar el puño (pulgar fuera)

Se crea un nodo cuyo propósito es que al ser ejecutado la Shadow Hand de la simulación mueva los dedos siguiendo un orden y algunos de forma simultánea para cerrar el puño.

El propósito es probar las articulaciones y su movimiento simultáneo.

6.5.1.2.5 Ejecutar un programa para abrir la mano (pulgar fuera)

Variación de la prueba anterior, cambiando el orden de movimiento de los dedos y la posición del pulgar.

[3] Shadow Dexterous Hand Technical Specification - Página 8: http://www.shadowrobot.com/wp-content/uploads/shadow_dexterous_hand_technical_specification_E1_20130101.pdf
[4] ROS.org - Shadow Robot: http://wiki.ros.org/shadow_robot

6.5.1.2.6 Ejecutar un programa para que la mano haga un signo

Se preparan las coordenadas de las articulaciones en distintos métodos de un nuevo nodo para que la mano adopte una posición neutra (mano abierta) para luego tomar la forma del signo y luego volver a abrirse.

Esta prueba se realiza con cada uno de los signos. Este programa servirá de base para la siguiente prueba, de ahí que todos los métodos para cada signo se vayan añadiendo al mismo.

6.5.1.2.7 Ejecutar un programa para que la mano realice una secuencia de signos

Lo mismo que el anterior, pero ahora debe realizar distintos signos de forma secuencial.

6.5.1.3 Reconocimiento de voz (Pocketsphinx)

6.5.1.3.1 Reconocer frase

Se utiliza Pocketsphinx con cualquier diccionario del que se tenga conocimiento de las frases que reconoce, y se prueban varias de ellas varias veces para comprobar si reconoce lo que se le dice.

6.5.2 Pruebas de Integración y del Sistema

6.5.2.1 Enviar y recibir mensajes entre el robot y la página web

6.5.2.1.1 Enviar mensaje al robot desde la página web

Activamos en el robot Roscore, Rosbridge y el nodo suscriptor del tutorial[5], que se suscribirá al mismo canal en el que publica la página web, y con la página web con la URL del robot, siendo localhost en las pruebas ya que ROS y la página web funcionaban sobre la misma máquina, se envían mensajes al robot y se espera que la ventana del nodo suscriptor del tutorial aparezcan los mismos mensajes que se le han enviado.

6.5.2.1.2 Iniciar escucha en el mismo canal de publicación

Una forma de saber al mismo tiempo si la página web puede enviar mensajes al robot y al mismo tiempo recibir mensajes del mismo, se suscribe la página web al mismo canal de escucha al que envía los mensajes. En el robot deben estar funcionando Roscore y Rosbridge. Desde la página se escriben los mensajes y estos mismos deben mostrarse en la misma, de ese modo se comprueba que han pasado por el robot. Si se cierra Roscore o Rosbridge, la página web dejaría de recibir los mensajes que envía, ya que no llegan al robot al faltar el núcleo o el enlace al mismo.

[5] Writing the Subscriber Node : http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber(python)

6.5.2.1.3 Enviar mensajes traducidos

Esta prueba extiende las pruebas anteriores. Se usará la versión para enviar traducciones de la página web, deben estar activos Roscore, Rosbridge y el nodo de suscripción del tutorial, el cual recibirá versiones traducidas de los mensajes enviados desde la página web. Si la página web se suscribe a su mismo canal de publicación también aparecerán los mensajes traducidos.

6.5.2.1.4 Traducir los mensajes recibidos

Esta prueba extiende de la anterior, pero los mensajes que se traducen son los recibidos, de modo que se utiliza otra versión de la página web, que traducirá los mensajes de entrada y no los de salida. En el robot se activan Roscore, Rosbridge y el nodo de suscripción del tutorial, desde la página web, que estará suscrita al mismo canal en el que escribe, se envían mensajes que aparecerán igual que como se envían en el nodo de suscripción, pero al recibirlos en la página web estos deberán aparecer traducidos.

Se hace así la prueba para demostrar que los mensajes se están enviando sin traducir y que la traducción solo se produce al recibir los mensajes.

6.5.3 Pruebas del sistema

6.5.3.1 Controlar el robot con el servicio web

6.5.3.1.1 Hacer hablar al robot desde la página web

Para esta prueba deben estar activos Roscore, Rosbridge, Sound Play y el nodo de habla, el cual debe estar suscrito al mismo canal en el que publica la página web. Tras escribir cualquier mensaje en la página web, el robot empezará a hablar por voz.

6.5.3.1.2 Hacer mover la Shadow Hand al robot desde la página web

Para esta prueba deben estar activos Roscore, Rosbridge, el nodo de control de la Shadow Hand, que estará suscrito al mismo canal de publicación de la página web; y la simulación de la Shadow Hand en Gazebo. Tras escribir cualquier mensaje en la página web, la mano virtual debería moverse acorde al mensaje recibido.

6.5.3.1.3 Recibir mensaje escuchado del robot

Para esta prueba deben estar activos Roscore, Rosbridge y Pocketsphinx, la página web tiene que estar suscrita al mismo canal en el que publica Pocketsphinx. Al hablar al robot, en la página web deben aparecer los mensajes reconocidos por Pocketsphinx.

6.5.3.1.4 Hacer hablar al robot desde la página web un mensaje traducido

Combinación de dos pruebas anteriores. Deben estar activos Roscore, Rosbridge, el nodo de habla y Sound Play, la página web usada debe ser la versión que envía mensajes traducidos. Se debe enviar desde la web un mensaje y la traducción de este será dicha por el robot.

6.5.3.1.5 Hacer mover la Shadow Hand al robot desde la página web con un mensaje traducido

Combinación de dos pruebas anteriores. Deben estar activos Roscore, Rosbridge, el nodo de control de la Shadow Hand y la virtualización en Gazebo de la Shadow Hand, la página web usada debe ser la versión que envía mensajes traducidos. Se debe enviar un mensaje desde la web y la Shadow Hand interpretará su traducción.

6.5.3.1.6 Recibir mensaje escuchado del robot y mostrarlo traducido

Para esta prueba deben estar activos Roscore, Rosbridge y Pocketsphinx, la página web tiene que estar suscrita al mismo canal en el que publica Pocketsphinx y ser la versión que traduce los mensajes. Los mensajes escuchados por Pocketsphinx deben mostrarse traducidos en la página web.

6.5.3.2 Controlar la Shadow Hand por voz

6.5.3.2.1 Traducir voz a lenguaje de signos

Para esta prueba deben estar activos Roscore, Pocketsphinx, el nodo de control de la Shadow Hand y la simulación en Gazebo de la Shadow Hand. El nodo de control de la Shadow Hand debe estar suscrito al canal de publicación de Pocketsphinx para que lo que este escuche sea interpretado por la Shadow Hand.

6.5.4 Pruebas de Usabilidad y Accesibilidad

6.5.4.1 Preguntas de caracter general

Indique la frecuencia de uso del ordenador
1. Todos los días 2. Casi a diario 3. Algunos días 4. Nunca o casi nunca
Indique la frecuencia de uso de aplicaciones para teléfonos móviles o tablets
1. Todos los días 2. Casi a diario 3. Algunos días 4. Nunca o casi nunca
¿Para qué utiliza el ordenador/móvil/tablet? (puede marcar varias)

1. Trabajo
2. Ocio
3. Solo documentación
4. Correo electrónico
5. Navegar por Internet
6. Comunicación (chats/mensajería instantánea/foros)

¿Ha utilizado alguna aplicación parecida?

1. Sí
2. Solo en parte
3. No

¿Qué valora más (mayor prioridad) en una aplicación de estas características)?

1. Fácil de usar
2. Bien guiado
3. Rápido
4. Completo
5. Que consuma pocos recursos

6.5.4.2 Actividades guiadas

Para esta actividad podemos usar a dos usuarios a la vez, uno usando el navegador y otro usando el robot, intercambiándose los puestos posteriormente. También se puede realizar con una única persona que pueda manejar el navegador a la vez que puede usar el robot y ver si su comportamiento es correcto.

Ambos usuarios pueden hablar idiomas distintos y que los mensajes sean traducidos en el proceso de comunicación.

No es necesaria la prueba de la Shadow Hand con usuarios ya que el objetivo de este proyecto es comprobar hasta qué grado la tecnología permite este tipo de comunicación, añadiendo que la simulación no funciona a tiempo real y la comunicación se haría lenta.

6.5.4.2.1 Usuario de la página web

Su objetivo es mantener una conversación con el usuario que utiliza el robot:

- Enviar mensajes al robot.
- Leer mensajes del robot.

Se le adjuntaría al usuario el siguiente cuestionario:

Facilidad de Uso	**Siempre**	**Frecuentemente**	**Ocasionalmente**	**Nunca**
¿El aspecto de la web le parece familiar en comparación con otras aplicaciones de comunicación?				
¿Es capaz de enviar mensajes?				
¿Es capaz de recibir mensajes?				
Funcionalidad	**Siempre**	**Frecuentemente**	**Ocasionalmente**	**Nunca**
¿Le parece correcta la traducción?				
¿El tiempo de respuesta de la aplicación es muy grande? (Esta pregunta solo en caso de que el usuario esté usando la web y el robot a la vez)				
Calidad del Interfaz				
Aspectos gráficos	**Muy Adecuado**	**Adecuado**	**Poco Adecuado**	**Nada Adecuado**
El tipo y tamaño de letra es				
Los colores empleados son				
Observaciones				
Cualquier comentario del usuario				

6.5.4.2.2 Usuario del robot

Su objetivo es mantener una conversación con el usuario que utiliza la página web:

- Escuchar mensajes hablados por el robot.
- Hablarle al robot para que envíe mensajes.

Facilidad de Uso	**Siempre**	**Frecuentemente**	**Ocasionalmente**	**Nunca**
¿El robot le oye cuando habla?				
¿Escucha bien el sonido del robot?				
¿Le resulta sencillo el uso del robot?				
Funcionalidad	**Siempre**	**Frecuentemente**	**Ocasionalmente**	**Nunca**
¿El robot entiende lo que dice?				
¿El tiempo que tarda el robot en reconocer lo que dice le parece el adecuado?				
Observaciones				
Cualquier comentario del usuario				

6.5.4.2.3 Cuestionario para el responsable de las pruebas

Aspecto Observado	Notas
Reacción del usuario frente a la aplicación/robot	
Tiempo de aprendizaje del usuario	
Errores leves cometidos	
Errores graves cometidos	
Problemas en el funcionamiento	
Comportamiento no esperado del usuario	
Comportamiento no esperado de la aplicación	
Otros problemas técnicos	

6.5.5 Pruebas de Rendimiento

Para traducir los mensajes que se envían y reciben del robot se utiliza un servicio externo, el traductor Frengly, de modo que nos interesa saber si es lo suficientemente rápido para la labor. Para esto se crea una página web especial que utilizará este servicio de la misma forma que se usa con el robot, pero en el que se mide cuánto tiempo se tarda en realizar las traducciones. A continuación se incluye el código del método que comprueba los tiempos de las traducciones:

```
  function traducir(){
  var p1 = new Date(); //Punto de inicio
    var texto = document.getElementById("caja").value; //Cogemos
el texto de caja

        var
xmlDoc=loadXMLDoc("http://syslang.com/?src=es&dest=en&text="+tex
to+"&email=dark_enzan@yahoo.es&password=bass14");
        var traduccion =
xmlDoc.getElementsByTagName("translation")[0].childNodes[0].node
Value + "<br>";

    document.getElementById('escuchado').innerHTML += texto
+"<br>"+ traduccion +""; //'Received message on ' +
listener.name + ': ' + message.data+'<br>'; //Escribimos el
nuevo mensaje en la página.
        document.getElementById('Boton').scrollIntoView();
//Bajamos el scroll automáticamente.

    var p2 = new Date(); //Punto final

    //El siguiente código sirve para medir cuánto tiempo se
tarda en realizar cada traducción:
    var segundos = 0;
    var milisegundos = 0;

    segundos = p2.getSeconds() - p1.getSeconds();
```

```
    if (segundos < 0){
        segundos = segundos + 60;
    }
    milisegundos = p2.getMilliseconds() - p1.getMilliseconds();
    if (milisegundos < 0){
        milisegundos = milisegundos + 1000;
    }

    document.getElementById('escuchado').innerHTML += segundos
+" segundos "+ milisegundos +" milisegundos<br><br>";

    //Fin de la medición

        document.getElementById("caja").value = '';
        document.getElementById("caja").submit();

  }
```

Capítulo 7. Implementación del Sistema

7.1 Estándares y Normas Seguidos

El estándar seguido a la hora de desarrollar este proyecto se ha definido por un sistema cíclico de incrementos compuesto de cuatro partes: Análisis, Diseño, Desarrollo y Pruebas.

Primero se analiza qué es lo que se necesita o se pretende alcanzar, investigando el funcionamiento de la tecnología y las posibles opciones, posteriormente se pasa al diseño del sistema, cómo debe funcionar y qué resultado se espera obtener. Tras esto se desarrolla el código y se procede a realizar las pruebas.

La fase de análisis puede comenzar sin la finalización del ciclo anterior y según el tipo y la naturaleza de la mejora esta se puede empezar incluso a diseñar antes de acabar el desarrollo y las pruebas de la anterior

7.2 Lenguajes de Programación

El lenguaje de programación usado en el desarrollo de los nodos de este proyecto es Python en su versión 2.7, que es la que se incluye con Ubuntu 12.04.

Python es un lenguaje de programación interpretativo, interactivo y orientado a objetos. Incorpora módulos, excepciones, escritura dinámica (dynamic typing, permite alterar el código con los programas en funcionamiento), tipos de datos dinámicos y clases. Dispone de interfaces a llamadas de sistema y librerías. Python es portable, funciona en Unix/Linux, Mac y PC con MS-DOS, Windows y OS/2.

Para que la página web pudiera establecer contacto con el robot se ha utilizado Javascript, que es un lenguaje de programación dinámico comúnmente usado en páginas webs y que permite implementar scripts para clientes para interactuar con el usuario, controlar el navegador, establecer comunicaciones asíncronas y modificar el contenido mostrado en la página; cosaws que se hacen en este proyecto.

7.3 Herramientas y Programas Usados para el Desarrollo

7.3.1 Robot Operating System (ROS)

Es un conjunto de librerías y herramientas para ayudar a construir aplicaciones para robots. Su objetivo es simplificar la labor para crear comportamientos complejos para una gran variedad de robots.

ROS ha sido creado para apoyar el desarrollo colaborativo de software robótico, permitiendo a cada desarrollador especializarse en un área determinada y poder usar el software de otros especialistas en otros áreas a la vez que comparte sus creaciones, creando apoyo entre los trabajos de los demás.

Las versiones de ROS utilizadas en este proyecto han sido Groovy Galapagos e Hydro Medusa, siendo esta la más reciente y la recomendada para probar el proyecto ya que es la versión en la que funciona la simulación de la Shadow Hand, si bien el resto de funciones se pueden probar en Groovy.

La versión Indigo Igloo iba a salir durante el mes de mayo de 2014, pero aún en julio no ha salido, de modo que no se ha podido comprobar su compatibilidad con el proyecto.

A continuación se pasan a detallar los componentes de ROS importantes para este proyecto:

7.3.1.1 Roscore

Roscore es un conjunto de nodos y programas requeridos en un sistema basado en ROS. Debe estar funcionando para que los nodos de ROS se comuniquen. Se lanza con el comando roscore, pero si se lanza un nodo con el comando roslaunch y no detecta un roscore, pondrá uno a funcionar.

7.3.1.2 Rosbridge

Rosbridge permite la comunicación con programas que no pertenezcan a ROS. Ofrece un servidor WebSocket para que las páginas web puedan interactuar con él.

7.3.1.3 Sound Play

Sound Play (sound_play) ofrece un nodo de ROS que convierte comandos de ROS en sonidos. Soporta sonidos pregrabados, reproduciendo archivos OGG y WAV, y también permite la síntesis de voz a través de Festival[18]. Viene con una voz Festival en inglés por defecto, pero permite la instalación de más voces, con soporte para distintos idiomas y la posibilidad de alternar entre ellas.

7.3.1.4 Pocketsphinx

Este paquete permite el acceso al sistema de reconocimiento de voz de Pocketsphinx. Requiere un modelo de lenguaje y un diccionario, que se puede conseguir utilizando la Online Sphinx Knowledge Base Tool v.3[6], permitiendo crear un diccionario acorde a nuestras necesidades.

7.3.1.5 Gazebo

Gazebo ROS Packages es un set de paquetes de ROS que otorgan las interfaces necesarias para simular un robot en el simulador Gazebo. Se integra con ROS utilizando sus mensajes, servicios y reconfiguración dinámica.

7.3.2 Oracle VM VirtualBox

VirtualBox es una máquina virtual de propósito general. En las máquinas utilizadas para el desarrollo del proyecto las versiones de Ubuntu anteriores a 14.04 no eran compatibles gráficamente, y esta versión de Ubuntu no dispone de una versión compatible de ROS, de modo que se esperaba que Indigo Igloo se publicase para utilizarla, pero como eso no ha ocurrido la alternativa ha sido utilizar máquinas virtuales de las versiones anteriores de Ubuntu, concretamente con la versión Ubuntu 12.04, pudiendo así usar ROS en estas máquinas, incluyendo dispositivos de las mismas como el micrófono, compatible en una máquina virtual de Ubuntu que funcionase en un sistema Ubuntu, no siendo el mismo caso en Windows.

7.3.3 Gedit

Es un editor de texto incluido en Ubuntu utilizado para el desarrollo de los nodos creados para este proyecto y para el desarrollo de la página web.

7.4 Creación del Sistema

7.4.1 Problemas Encontrados

7.4.1.1 Incompatibilidad con Ubuntu

Para este proyecto era necesario utilizar Ubuntu, ya que es sobre este sistema operativo donde funciona ROS, en los ordenadores usados las versiones anteriores a 14.04 no funcionaban con GUI, elemento necesario para este proyecto, y al intentar instalar los drivers directamente dejaban de funcionar. Se han perdido varios días de desarrollo en probar distintas versiones de Ubuntu. La versión 14.04 no podía usarse para el desarrollo ya que no

[6] Sphinx Knowledge Base Tool -- VERSION 3: http://www.speech.cs.cmu.edu/tools/lmtool-new.html

había una versión compatible de ROS con dicho sistema operativo, a la espera de Indigo Igloo, la cual se retrasó en su salida.

7.4.1.1.1 Solución

La solución llegó de la mano de la virtualización con la Oracle VM VirtualBox, que permitía instalar y utilizar las versiones anteriores de Ubuntu.

7.4.1.2 Funcionamiento del micrófono

Utilizando una máquina virtual de Ubuntu en Windows no permite utilizar el micrófono, esto impedía las pruebas con Pocketsphinx.

7.4.1.2.1 Solución

La solución se consiguió al exportar dichas máquinas virtuales que se usaban en Windows e importarlas en las máquinas virtuales usadas en Ubuntu 14.04, de esta forma al ser el sistema operativo raíz y el virtualizado distintas versiones del mismo sistema operativo, se podía utilizar el micrófono ya que este accedía a las mismas zonas de memoria en ambos sistemas.

7.4.1.3 Desaparición de la Shadow Hand virtualizada

En la simulación de la Shadow Hand en ocasiones se produce un fallo al colisionar algunas partes de la misma que produce que la mano desaparezca de la simulación.

7.4.1.3.1 Solución

La solución pasa por devolver la mano siempre a una posición neutral y establecer unos tiempos de movimiento para que al hacer un signo o restaurar la posición de tiempo a las articulaciones a moverse sin producirse una colisión grave entre los miembros.

Estos tiempos pueden variar según el equipo en el que se haga la simulación, ya que no todos pueden hacer una virtualización a tiempo real, de modo que en el nodo existen unos atributos para definir estos tiempos y que se pueden aumentar o disminuir según dónde se simule la mano.

7.4.1.4 Reconocimiento de voz

Se ha intentado usar en este proyecto el reconocimiento de voz de Google, con un alto grado de acierto y compatible con múltiples idiomas y acentos, mediante el uso de diversas API de código abierto disponibles para Python, pero estas no funcionaban debido a que Google cambia cada cierto tiempo el protocolo de acceso, de modo que se vuelven inútiles.

7.4.1.4.1 Solución

Al no poderse usar el sistema de reconocimiento de voz de Google, se ha optado por no incluir esta alternativa al Pocketsphinx.

7.4.2 Descripción Detallada de las Clases

Las clases que aquí se muestran son las desarrolladas para este proyecto, las cuales disponen de distintas versiones para comunicarse con distintos canales de escucha o publicación, según la necesidad, pero que en la base son lo mismo con una diferencia de pocas líneas de código:

7.4.2.1 Nodos de habla

Nombre	**Descripción**	**Dependencias**
escuchaWeb.py / escuchaWebEsp.py	Es el nodo que indica al robot qué debe decir y con qué voz ha de hacerlo.	roslib rospy std_msgs.msg (String) sound_play.msg (SoundRequest) sound_play. libsoundplay (SoundClient)
Responsabilidades		
Número	**Descripción**	
1	Escuchar en el canal de publicación de la página web.	
2	Preparar el mensaje para que Sound Play lo reproduzca con voz.	
3	Indicar a Sound Play qué voz debe utilizar para reproducir el mensaje.	
Métodos		
Nombre	**Parámetros y tipos**	
sleep	t (int)	
callback	data (String)	
listener		
Atributos		
No tiene atributos.		
Observaciones		

7.4.2.2 Nodos de control de la Shadow Hand

No hay tipos propiamente dicho en los atributos, pero mantengo el apartado en la tabla para incluir el propósito del atributo.

Nombre	**Descripción**	**Dependencias**
controlSH.py	Es la clase que controla la posición de las articulaciones de la mano robótica.	roslib rospy time std_msgs.msg (Float64) pr2_controllers_msgs.msg (JointControllerState)
Responsabilidades		
Número	**Descripción**	
1	Escuchar del canal de publicación de la página web o del Pocketsphinx.	
2	Enviar las órdenes a la Shadow Hand (o su simulación) para representar	

	los signos del mensaje recibido.

Métodos

Nombre	Parámetros y tipos
cierra	
abre	
abre2	
abre3	
a	
b	
c	
[...]	
z	
mover	
listener	
main	

Atributos

Tipo o Clase	Nombre
int	tiempo_mucho
int	tiempo_medio
int	tiempo_poco
int	tiempo_poquisimo
Publisher	pubSass
Publisher	pubFfj3
Publisher	pubFfj0
Publisher	pubFfj4
Publisher	pubMfj4
Publisher	pubMfj3
Publisher	pubMfj0
Publisher	pubRfj4
Publisher	pubRfj3
Publisher	pubRfj0
Publisher	pubLfj5
Publisher	pubLfj4
Publisher	pubLfj3
Publisher	pubLfj0
Publisher	pubThj5
Publisher	pubThj4
Publisher	pubThj3
Publisher	pubThj2
Publisher	pubThj1

Observaciones

Se sobreentiende que hay un método por cada letra del alfabeto, cada uno destinado a que la mano tome la posición del signo de la letra a representar.

Los atributos de tiempo son constantes para el control de la mano simulada. La mano en la simulación no funciona a la velocidad normal, y según el equipo en el que se pruebe funciona más rápido o más despacio, con lo cual es necesario adaptar estas variables según en dónde se simule para darle tiempo a hacer los signos y a restaurar su posición.
Si no se hace esta adaptación las partes de la mano podrían colisionar de forma crítica y producir un error que haría desaparecer la mano de la simulación.

Los Publisher se podrían definir como objetos para el control de las articulaciones. Cada uno

está dirigido a una articulación distinta.

7.4.2.3 Javascript de la página web

Nombre	**Descripción**
saluda.html	Javascript para comunicar la página web con el robot.
Responsabilidades	
Número	**Descripción**
1	Enviar mensajes al robot.
2	Recibir mensajes del robot.
3	Traducir los mensajes a enviar y los que se reciban (opcional).
Métodos	
Nombre	
habla	
escucha	
invocaEscuchar	
traducirEsEn	
traducirEnEs	
pulsaIntro	
Atributos	
Tipo o Clase	**Nombre**
var	**ros**
var	**mensajeAnterior**
var	**estaEsperando**
var	**listener**
Observaciones	
Hay dos versiones de la página web, según si se envían y reciben los mensajes tal cual o si se traducen, siendo el código extra interno a los métodos.	

Capítulo 8. Desarrollo de las Pruebas

8.1 Pruebas Unitarias

En esta sección no solo se incluyen las pruebas, sino que se incluyen los pasos de preparación para realizar las pruebas:

8.1.1 Síntesis de voz

8.1.1.1 Sintetizar voz

8.1.1.1.1 Preparación previa

Se va a probar si funciona el Sintetizador de voz en el sistema. Primero instalamos audio_common y sound_play:

```
sudo apt-get install ros-hydro-audio-common
rosdep install sound_play
rosmake sound_play
```

8.1.1.1.2 Prueba

Si no estamos usando Hydro Medusa, sustituimos "hydro" por el nombre de la distribución que estemos usando. Luego iniciamos roscore en una ventana de terminal:

```
roscore
```

Posteriormente en otra ventana de terminal iniciamos soundplay:

```
rosrun sound_play soundplay_node.py
```

Si todo ha ido bien veremos un mensaje de que Sound_play está listo para reproducir sonido. Como ahora solo tiene la voz por defecto probaremos a que diga algo en inglés desde una nueva ventana de terminal (la tercera):

```
rosrun sound_play say.py "hello my lady, hello my baby"
```

Si todo ha salido bien, se escuchará la frase.

8.1.1.2 Sintetizar voz en castellano

8.1.1.2.1 Preparación previa

Ahora que podemos sintetizar voz, nos interesa sintetizarla en castellano. Con el siguiente código podemos ver las voces Festival disponibles:

```
sudo apt-cache search --names-only festvox-*
```

Encontramos la voz masculina festvox-ellpc11k, procederemos a su instalación:

```
sudo apt-get install festvox-ellpc11k
```

Para comprobar que se ha instalado y con qué nombre debemos referirnos para usar esta voz usaremos:

```
ls /usr/share/festival/voices/spanish
```

Veremos que la voz instalada se llama el_diphone.

8.1.1.2.2 Prueba

Al igual que en la práctica anterior debemos tener en una terminal roscore, en otra sound_play y en la tercera escribiremos la siguiente línea de código para comprobar que la voz funciona:

```
rosrun sound_play say.py "En un lugar de la Mancha, de cuyo
nombre no quiero acordarme" voice_el_diphone
```

Si lo hemos hecho todo bien, se escuchará la frase.

8.1.1.2.3 Problema encontrado

Cuando el texto incluye letras como la ñ o vocales con tilde, deletreará la palabra en la que se encuentra dicha letra y la sustituirá por x.

8.1.2 Movimiento de la Shadow Hand (mano robótica)

8.1.2.1 Mover las articulaciones de los dedos

8.1.2.1.1 Preparación

Para esta prueba vamos a necesitar instalar los datos de la simulación de la Shadow Hand e usar su simulación con Gazebo. Las instrucciones se pueden encontrar en el apartado sobre Shadow Robot de la wiki de ROS[7].

Primero se procede a instalar el software para la simulación de la Shadow Hand:

```
sudo apt-get install ros-hydro-shadow-robot
```

Una vez instalado, iniciamos la simulación con el siguiente comando:

```
roslaunch sr_hand gazebo_arm_and_hand.launch
```

8.1.2.1.2 Prueba

Probamos a mover las distintas articulaciones de la Shadow Hand, para conocerlas usamos la Shadow Dexterous Hand Technical Specification[8], donde disponemos de la siguiente tabla, además de diversos gráficos que muestran a qué engranaje se refiere cada nombre:

Joint(s)	Degrees		Radians		Notes
	Min	Max	Min	Max	
FF1, MF1, RF1, LF1	0	90	0	1.571	Coupled
FF2, MF2, RF2, LF2	0	90	0	1.571	
FF3, MF3, RF3, LF3	0	90	0	1.571	
FF4, MF4, RF4, LF4	-20	20	-0.349	0.349	
LF5	0	45	0	0.785	
TH1	0	90	0	1.571	
TH2	-25	25	-0.436	0.436	
TH3	-12	12	-0.209	0.209	
TH4	0	70	0	1.222	
TH5	-55	55	-0.960	0.960	
WR1	-45	30	-0.785	0.524	
WR2	-28	8	-0.489	0.140	

Figura x.x Engranajes de la Shadow Hand y sus limitaciones

Hacemos pruebas con códigos como el siguiente:

[7] ROS.org - shadow_robot: http://wiki.ros.org/shadow_robot
[8] Shadow Dexterous Hand Technical Specification

```
rostopic pub /sh_x_position_controller/command std_msgs/Float64
y
```

Sustituimos la **x** por el nombre de la articulación, que puede ser uno de los siguientes: wrj1, wrj2, ffj4, ffj3, ffj0, mfj4, mfj3, mfj0, rfj4, rfj3, rfj0, lfj5, lfj4, lfj3, lfj0, thj5, thj4, thj3, thj2, thj1. Sustituimos la **y** por los radianes de la posición a adoptar.

NOTA: En los dedos, salvo en el pulgar, las articulaciones 1 y 2 no son independientes, y se mueven con la orden numerada como 0.

8.1.2.2 Ordenar mover las articulaciones de los dedos fuera de sus límites

Se puede hacer durante la prueba anterior. Simplemente se comprueba que las articulaciones se detienen al alcanzar su límite.

8.1.2.3 Ejecutar un programa para cerrar el puño (pulgar dentro)

Esta prueba se realiza al comienzo de la creación del programa de interpretación, de modo que el código a utilizar se puede ver en la sección de código. El método a ejecutar para esta prueba es cerrar(), que es una posición de la mano en la que el puño está cerrado con el pulgar dentro.

Para probarlo con el código entregado habría que sustituir el código del main por el siguiente:

```
if __name__ == '__main__':
    rospy.init_node('joints_link_test', anonymous=True)
    time.sleep(3)
    abrir3()
    cerrar()
```

El método abrir3 pone la mano en una posición neutral (ya que al iniciar la simulación los dedos están torcidos), dejando un tiempo prudencial para que esto se produzca. Posteriormente se ejecuta a(), cerrando primero los dedos superiores de la mano y moviendo el pulgar al final.

Si ocurre algún fallo o no se consigue la posición deseada, significa que la simulación va muy despacio y se deben aumentar los valores de los atributos de tiempo del programa para ese ordenador.

8.1.2.4 Ejecutar un programa para que la mano haga un signo

Aquí se sigue con la prueba anterior, que se realizará tantas veces como letras se vayan a probar, de modo que la prueba se va haciendo mientras se implementan los métodos, con un código en el main similar al siguiente, donde se sustituye "metododelaletra" por el nombre del método correspondiente (a, b, c, d, etc.) una vez desarrollado:

```
if __name__ == '__main__':
```

```
    rospy.init_node('joints_link_test', anonymous=True)
    time.sleep(3)
    abrir3()
    metododelaletra()
```

8.1.2.5 *Ejecutar un programa para que la mano realice una secuencia de signos*

Una vez acabados todos los signos, podemos representar una secuencia de signos. Se puede usar la aplicación terminada, pero cuando no estaba terminada se hacía una secuencia en el main:

```
if __name__ == '__main__':
    rospy.init_node('joints_link_test', anonymous=True)
    time.sleep(3)
    abrir3()
    h()
    abrir3()
    o()
    abrir3()
    l()
    abrir3()
    a()
```

8.1.3 Reconocimiento de voz (Pocketsphinx)

Primero se instala Pocketsphinx desde una terminal:

```
sudo apt-get install ros-hydro-pocketsphinx
```

Posteriormente nos bajaremos los archivos del tutorial, que en realidad es un programa para hacer pruebas basado en los comandos de la Robocup, una competición de robots:

```
svn checkout http://pi-robot-ros-
pkg.googlecode.com/svn/trunk/pi_tutorials/pi_speech_tutorial
rosmake --rosdep-install pi_speech_tutorial
```

De esta forma ya tenemos un diccionario con el que hacer pruebas, así que iniciamos Pocketsphinx:

```
roslaunch pocketsphinx robocup.launch
```

Al haber usado roslaunch también se inicia Roscore. Las frases y palabras que reconoce están relacionadas con un domicilio y utensilios, como suele ser en las Robocup, de modo que podremos probar frases como las siguientes:

- Go to the sofa
- Bring me the spoon
- Kitchen
- Lemon

8.2 Pruebas de Integración y del Sistema

Caso de Uso 1.1: Enviar y recibir mensajes entre el robot y la página web	
Prueba	**Resultado Esperado**
Enviar mensaje al robot desde la página web	El mensaje llega al robot, reflejándose en el Roscore.
	Resultado Obtenido
	El mensaje llega al robot, aparece en Roscore. Al combinar esta prueba con la siguiente, el mensaje también aparecerá de vuelta en la página web.
Prueba	**Resultado Esperado**
Iniciar escucha en el mismo canal de publicación	Al enviar un mensaje, este pasará por el robot y volverá a la página web.
	Resultado Obtenido
	Al enviar el mensaje, este se mostró en la parte superior de la página web ya que estaba escuchando en el mismo canal del robot en el que publicaba, probando simultáneamente la publicación y la escucha del robot.
Prueba	**Resultado Esperado**
Enviar mensajes traducidos	Al enviar un mensaje este llegará traducido.
	Resultado Obtenido
	Al enviar un mensaje, en el robot aparece traducido, y en la página web, siendo el canal de escucha el mismo de publicación, también se ve traducido.
Prueba	**Resultado Esperado**
Traducir los mensajes recibidos	Al recibir un mensaje, éste se mostrará traducido.
	Resultado Obtenido
	Al enviar un mensaje al robot, este aparece sin traducir en él, pero en la página web, suscrita al canal de publicación, se muestra traducido ya que es al recibirlo de vuelta cuando se traduce.

Caso de Uso 2.1: Controlar al robot con el servicio web	
Prueba	**Resultado Esperado**
Hacer hablar al robot desde la página web.	El robot reproduce el mensaje enviado.
	Resultado Obtenido
	El robot reproduce el mensaje enviado.
Prueba	**Resultado Esperado**
Hacer mover la Shadow Hand al robot desde la página web	La mano interpreta el mensaje enviado.
	Resultado Obtenido
	La mano interpreta el mensaje enviado. El sistema falla al recibir signos como vocales con tildes o 'ñ', pero sigue funcionando.
Prueba	**Resultado Esperado**

Recibir mensaje escuchado del robot	Se recibe en la web en texto el mensaje escuchado del robot.
	Resultado Obtenido
	El robot interpreta el mensaje escuchado y esta interpretación en texto se envía a la página web. No siempre se reconoce exactamente lo que se le dice e interpreta ruido como palabras. Exige una correcta calidad en la pronunciación.
Prueba	**Resultado Esperado**
Hacer hablar al robot desde la página web un mensaje traducido	Se envía un mensaje y el robot debe reproducir el mensaje traducido.
	Resultado Obtenido
	El robot reproduce el mensaje que se ha enviado desde la web traducido. La traducción es muchas veces muy literal.
Prueba	**Resultado Esperado**
Hacer mover la Shadow Hand al robot desde la página web con un mensaje traducido	Se envía un mensaje y la Shadow Hand interpreta la traducción.
	Resultado Obtenido
	El robot recibe el mensaje y la mano interpreta la traducción. La traducción es muchas veces muy literal. El sistema falla al recibir signos como vocales con tildes o 'ñ', pero sigue funcionando.
Prueba	**Resultado Esperado**
Recibir mensaje escuchado del robot y mostrarlo traducido	La página web recibe un mensaje del robot y lo muestra traducido.
	Resultado Obtenido
	El robot interpreta el mensaje escuchado y esta interpretación en texto se envía a la página web, donde se muestra traducido. No siempre se reconoce exactamente lo que se le dice e interpreta ruido como palabras. Exige una correcta calidad en la pronunciación. La traducción es muchas veces muy literal.

Caso de Uso 2.2: Controlar la Shadow Hand por voz	
Prueba	**Resultado Esperado**
Traducir voz a lenguaje de signos	La mano se mueve según la interpretación de lo escuchado.
	Resultado Obtenido
	La mano se mueve según la interpretación de lo escuchado. No siempre se reconoce exactamente lo que se le dice e interpreta ruido como palabras. Exige una correcta calidad en la pronunciación.

Ejecutamos las pruebas funcionales ya diseñadas anteriormente y anotamos el resultado que obtenemos, comparándolo con el que especificamos anteriormente. Podemos hacerlo a partir de una tabla modificada de la anterior como ésta (si es más sencillo, puede hacerse con pequeñas tablas independientes para cada caso):

Caso de Uso 1.1: Añadir Usuario	
Prueba	**Resultado Esperado**
Añadir un usuario no existente	El sistema posee un usuario más
	Resultado Obtenido
	El sistema efectivamente posee un usuario más
Prueba	**Resultado Esperado**
Añadir un usuario que ya existe	El sistema no posee un usuario más y se muestra un dialogo notificándolo
	Resultado Obtenido
Prueba	**Resultado Esperado**
Cancelar la Operación	El sistema permanece sin cambios.
	Resultado Obtenido

8.3 Pruebas de Rendimiento

A continuación se muestran los resultados de rendimiento del servicio de traducción:

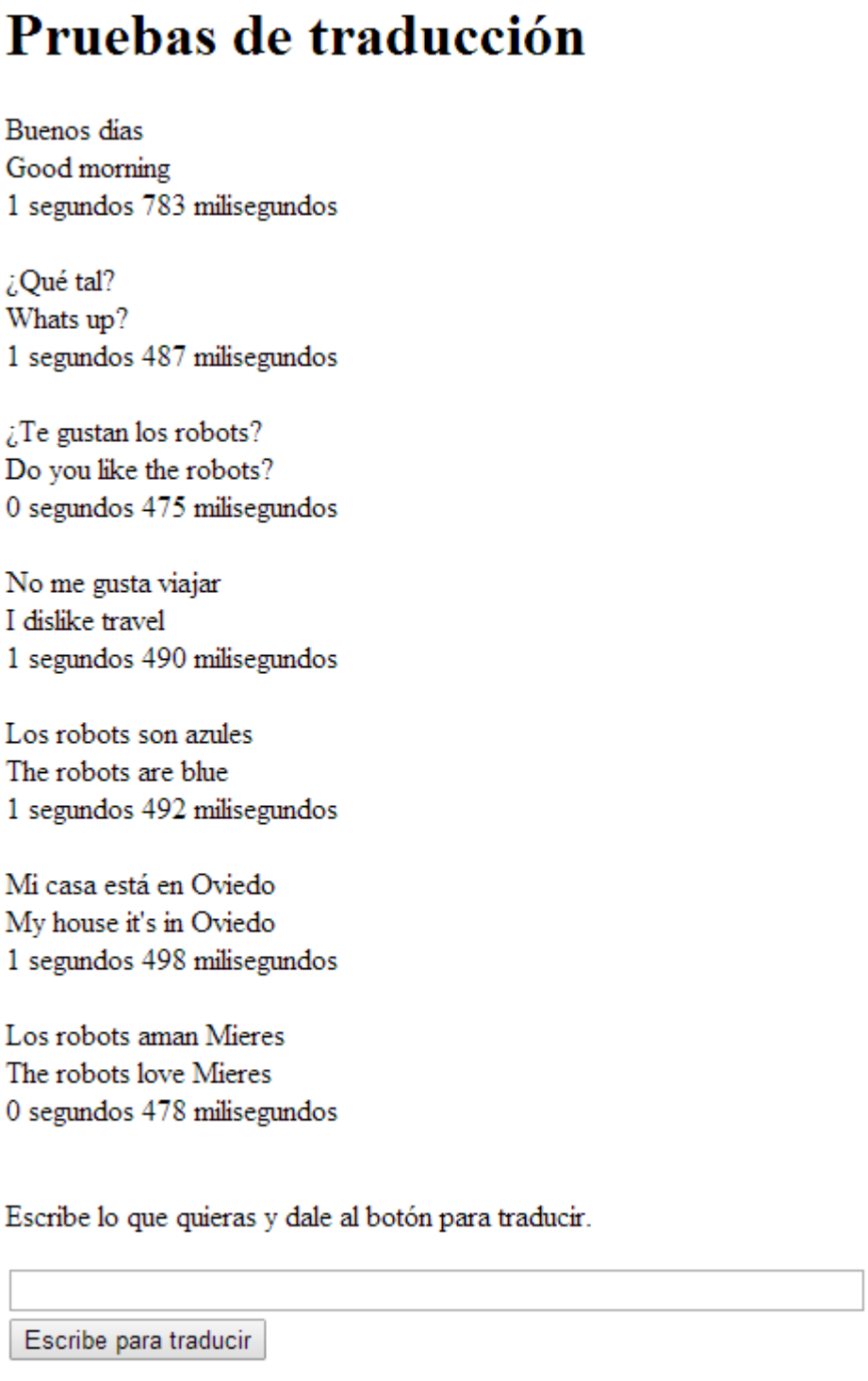


Figura x.x Prueba de rendimiento del traductor

Texto originalT	Traducción	Tiempo (milisegundos)

Buenos días	Good morning	1783
¿Te gustan los robots?	Do you like the robots?	475
No me gusta viajar	I dislike travel	1490
Los robots son azules	The robots are blue	1492
Mi casa está en Oviedo	My house it's in Oviedo	1498
Los robots aman Mieres	The robots love Mieres	478

La media que obtenemos para cada traducción es de 1202 milisegundos en cada traducción, lo cual es un tiempo aceptable ya que permite una comunicación fluida.

Capítulo 9. Manuales del Sistema

9.1 Manual de Instalación

9.1.1 Preparación del sistema

Para probar los elementos del proyecto, es necesario instalar ROS y los módulos necesarios. Este proyecto solo funciona en Ubuntu, habiéndose usado para este proyecto las versiones 12.04 de Ubuntu y la versión Hydro Medusa de ROS. Algunas partes de este proyecto pueden funcionar en otras versiones de ROS, pero no están del todo probadas para estos casos y podrían necesitar pasos extra. Los pasos de instalación de ROS se pueden encontrar más detallados en el tutorial[9] de su página web. ROS Hydro Medusa solo funciona en Ubuntu 12.04, 12.10 y 13.04. En caso de usar otra versión de Ubuntu se recomienda usar otra versión de ROS compatible o usar máquinas virtuales.

1. Permitir que Ubuntu acepte contenidos universe, restricted y multiverse. Para ello accedemos al Centro de Software de Ubuntu, vamos a la pestaña Editar y elegimos Orígenes de Software. Si no estuviera alguna marcada, la marcamos:

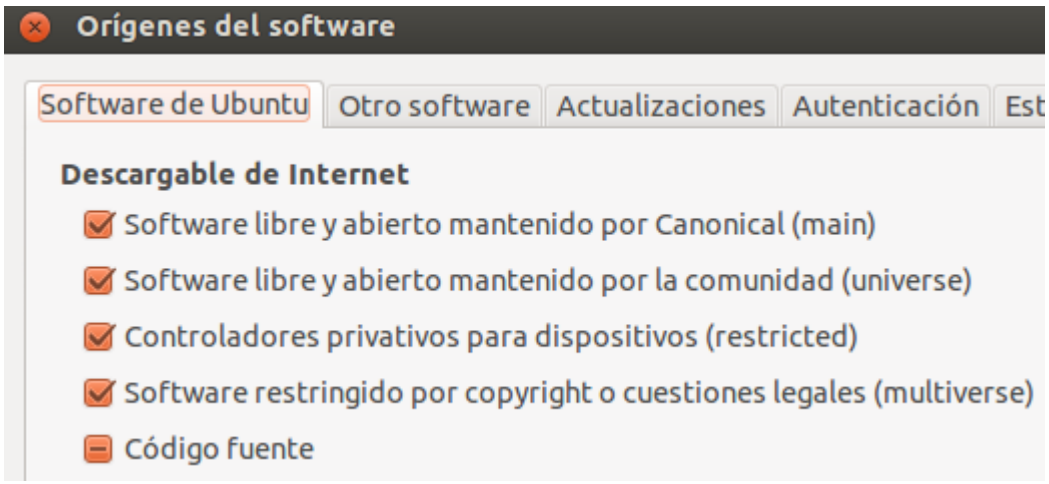


Figura x.x Orígenes del software de Ubuntu

2. Preparar los repositorios de Ubuntu desde una ventana de terminal con el siguiente código (según la versión de Ubuntu que se esté utilizando):

    a. Ubuntu 12.04 (Precise)

```
• sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu
  precise main" > /etc/apt/sources.list.d/ros-latest.list'
```

9

b. Ubuntu 12.10 (Quantal)

```
• sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu
  quantal main" > /etc/apt/sources.list.d/ros-latest.list'
```

c. Ubuntu 13.04 (Raring)

```
• sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu
  raring main" > /etc/apt/sources.list.d/ros-latest.list'
```

3. Preparar las claves:

```
• wget
  https://raw.githubusercontent.com/ros/rosdistro/master/ros
  .key -O - | sudo apt-key add -
```

4. Actualizar las librerías:

```
• sudo apt-get update
```

5. Hacer la instalación completa de escritorio. El tiempo variará según las capacidades del ordenador y la calidad de la conexión:

```
• sudo apt-get install ros-hydro-desktop-full
```

6. Una vez instalado, iniciar rosdep:

```
• sudo rosdep init
• rosdep update
```

7. Instalar Rosinstall:

```
• sudo apt-get install python-rosinstall
```

8. Instalar Pocketsphinx:

```
• sudo apt-get install ros-hydro-pocketsphinx
```

9. Instalar la simulación de la Shadow Hand (si no usásemos ROS Hydro o una versión posterior, sería necesario instalar Gazebo):

```
• sudo apt-get install ros-hydro-shadow-robot
```

10. Instalar Rosbridge:

```
• sudo apt-get install ros-hydro-rosbridge-server
```

11. Instalar Audio Common, Sound Play y la voz en castellano "el_diphone":

```
• sudo apt-get install ros-hydro-audio-common
• rosdep install sound_play
• rosmake sound_play
• sudo apt-get install festvox-ellpc11k
```

NOTA: Si el último paso no funciona debido a que la voz no está disponible, volver al paso 8.1.1.2.1 y seguir las instrucciones para instalar otra voz deseada.

9.1.2 Preparación del proyecto

1. Creamos el Workspace. Desde una terminal escribimos lo siguiente:

```
• mkdir ~/proyecto/
• rosws init ~/proyecto/ /opt/ros/hydro
• source ~/proyecto/setup.bash
• mkdir ~/proyecto/sandbox
• rosws set ~/proyecto/sandbox
```

2. Crear un paquete de ROS:

```
• roscd
• cd sandbox
• roscreate-pkg interprete std_msgs rospy roscpp roslib
  diagnostic_msgs pocketsphinx sound_play
```

3. Construir un paquete de ROS:

```
• rosmake interprete
```

4. Copiamos en el directorio la carpeta scripts, que contiene los archivos ".py" del proyecto.

5. Copiamos en la carpeta intérprete las carpetas config y launch.

NOTA: Si no se están usando los mismos nombres para los directorios, es posible que tengamos que entrar en launch y modificar el archivo para que coincidan.

9.1.3 Preparación de las páginas web

Si las páginas web se van a probar en la misma máquina que ejecuta el robot, no hace falta cambiar nada, ya que están preparados para Localhost. En caso de que se quiera probar en una máquina externa o colocar en un servidor, es necesario cambiar la siguiente parte, marcada en rojo, por la IP del robot, referida a la variable ros para dirigirse al robot:

```
var ros = new ROSLIB.Ros({
  url : 'ws://localhost:9090'
});
```

Para ver la IP de la máquina escribimos en una terminal ifconfig:

```
daniel@daniel-VirtualBox:~/proyecto/sandbox$ ifconfig
eth0      Link encap:Ethernet  direcciónHW 08:00:27:59:97:34
          Direc. inet:10.0.2.15  Difus.:10.0.2.255  Másc:255.255.255.0
```

Figura x.x Cómo aparece la IP en Linux

La IP será la dirección inet de eth0 (siendo este el principal cable de red, puede haber más). En este ejemplo, el código de la página web quedaría así:

```
var ros = new ROSLIB.Ros({
  url : 'ws://10.0.2.15:9090'
});
```

9.2 Manual de Ejecución

9.2.1 Iniciar la comunicación entre la página web y el robot

Es necesario lanzar Rosbridge, que se puede hacer con el siguiente comando en una terminal:

- `roslaunch rosbridge_server rosbridge_websocket.launch`

De esta forma no solo lanzamos Rosbridge, también iniciamos el Roscore, por eso esto está al principio de este manual.

9.2.2 Iniciar el sonido

Con Roscore iniciado, en otra ventana de Terminal escribimos lo siguiente:

- `rosrun sound_play soundplay_node.py`

9.2.3 Iniciar el nodo de habla

En una nueva terminal, fijamos el directorio del proyecto como fuente y posteriormente iniciamos el nodo de habla, habiendo iniciado los nodos de los apartados 9.2.1 y 9.2.2:

```
• source ~/proyecto/setup.bash
```

Para que se escuche la voz en inglés:

```
• rosrun interprete escuchaWeb.py
```

Para que se escuche la voz en castellano:

```
• rosrun interprete escuchaWebEsp.py
```

NOTA-1: Si se han usado otros nombres para crear los directorios de trabajo, deben cambiarse estos nombres en los comandos que se escriban.

NOTA-2: Si no se ha instalado la voz "el_diphone" pero se ha instalado otra de interés, sustituir en escuchaWebEsp.py el texto referente a la misma por el nombre de la nueva voz que se vaya a utilizar.

9.2.4 Iniciar el reconocimiento de voz

Para usar Pocketsphinx con el diccionario incluido en el presente proyecto, en una nueva terminal se debe escribir el siguiente comando:

```
• roslaunch interprete comandos.launch
```

NOTA: Si se está usando una máquina virtual de Ubuntu sobre Windows, no funcionará el micrófono y, por tanto, no se podrá utilizar el reconocimiento de voz. Esto solo funcionará en una máquina real con Ubuntu o en una máquina virtual con Ubuntu que funcione bajo una máquina real con Ubuntu, no siendo necesario que la máquina real tenga instalado ROS.

9.2.5 Iniciar el nodo de control de la Shadow Hand

En una ventana de terminal, para iniciar el nodo de control de la Shadow Hand desde la página web escribimos:

```
• source ~/proyecto/setup.bash
• rosrun interprete controlSH.py
```

En caso de que queramos controlar la mano usando el reconocimiento de voz:

```
• source ~/proyecto/setup.bash
• rosrun interprete controlSHVoz.py
```

9.2.6 Iniciar la simulación de la Shadow Hand

Para iniciar la simulación de la mano robótica, en una terminal nueva escribimos lo siguiente:

```
• roslaunch sr_hand gazebo_arm_and_hand.launch
```

9.3 Manual de Usuario

El presente proyecto dota al robot de las capacidades de establecer una comunicación por voz con otra persona que lo hará a través de texto, permitiendo establecer por voz una comunicación con alguien que hable otro idioma o que no pueda escucharle o hablar, incluso a distancia.

9.3.1 Instrucciones para el uso de la página web

La página web dispone de varias versiones, una que envía y recibe los textos sin modificación y otra que traduce los textos que se envían y se reciben. No obstante, su manejo es igual en ambos casos, ya que las traducciones son automáticas y se realizan antes de mostrar o enviar los mensajes sin intervención del usuario.

El funcionamiento es muy similar al que se puede encontrar en un chat, en la parte superior aparecen los mensajes recibidos del robot, mientras que en la parte inferior hay una caja de texto para escribir y enviar los mensajes.

NOTA: Una limitación del robot es la pronunciación de signos de apertura, vocales con tilde y la ñ, evite su utilización en los mensajes en caso de escribir en castellano.

Para iniciar la escucha del robot debe pulsarse el botón superior, lo cual inicia el procedimiento de escucha que hace que muestre los mensajes interpretados por el robot.

Para enviar los mensajes se puede usar el botón para enviar mensajes al robot o también se puede presionar la tecla Intro.

Figura x.x Interfaz de la página web.

9.3.2 Instrucciones para el uso del robot

9.3.2.1 Comunicación con la página web

El robot reproducirá por voz de forma automática los mensajes recibidos desde la página web.

Si está activado el nodo de control de la Shadow Hand (mano robótica) para los mensajes recibidos de la página web y su simulación, podrá ver una aproximación de la interpretación del lenguaje de signos si el robot dispone de pantalla.

Puede comunicarse con el robot hablándole alto y claro a una distancia cercana al micrófono del mismo. El reconocimiento de voz funciona solo en inglés.

9.3.2.2 Voz a lengua de signos

Hable alto y claro a una distancia cercana al micrófono del mismo. El reconocimiento de voz funciona solo en inglés. Si está activado el nodo de control de la Shadow Hand (mano robótica) para los mensajes por voz y su simulación, podrá ver una aproximación de la interpretación del lenguaje de signos si el robot dispone de pantalla.

9.4 Manual del Programador

9.4.1 Cambiar el diccionario del reconocimiento de voz

Para ampliar el diccionario del reconocimiento de voz de Pocketsphinx o cambiarlo por uno que se adapte a nuestras necesidades hay que seguir los siguientes pasos:

1. Escribir un documento ".txt" con las palabras sueltas y frases sencillas que queramos incluir. Un ejemplo del formato sería el siguiente:

```
HELLO
HI
```

```
HOW ARE YOU
FINE
GOODBYE
I AM FINE
I'M FINE
GET READY
ROBOT
TALKING ROBOT
```

2. Cargamos el archivo en la Sphinx Knowledge Base Tool[10].

3. Obtendremos una serie de archivos. Podemos renombrarlos si queremos, pero al mismo nombre y respetando la extensión. Estos archivos hay que introducirlos en el directorio config.

4. Prepararemos un lanzador para estos archivos, dándole el nombre que le hemos dado a los archivos seguido de la extensión ".launch".

5. El contenido del archivo será el siguiente, adaptándolo a los nombres que le hayamos dado al diccionario sustituyendo comandos.lm y comandos.dic por los nombres correspondientes y, si no hubiésemos usado los nombres de los paquetes recomendados en el tutorial, también habría que cambiarlos:

```
<launch>
  <node name="recognizer" pkg="pocketsphinx" type="recognizer.py"
output="screen">
    <param name="lm" value="$(find interprete)/config/comandos.lm"/>
    <param name="dict" value="$(find
interprete)/config/comandos.dic"/>
  </node>
</launch>
```

6. Para usar el nuevo diccionario usaremos en una terminal nueva el siguiente código, sustituyendo **comandos** por el nuevo nombre que le hayamos dado:

- `roslaunch interprete comandos.launch`

9.4.2 Cambiar el servicio de traducción

Existen dos métodos para la traducción de textos en las páginas de traducción, es en éstos métodos donde se procesan los textos antes de enviarlos o recibirlos, de modo que solo habría que modificar el código de estos métodos para cambiar el servicio de traducción en caso de querer usar otro, ya sea gratuito o de pago.

[10] Sphinx Knowledge Base Tool: http://www.speech.cs.cmu.edu/tools/lmtool-new.html

9.4.3 Modificar el manejo de la Shadow Hand

Cada uno de los movimientos del nodo de control de la Shadow Hand se divide en métodos, y existe un objeto para cada una de las articulaciones. El programa fue creado basándose en el diccionario dactilológico español, pero existen algunas diferencias en algunos signos con otros idiomas, de modo que si se pretende modificar dichos signos bastaría con modificar los métodos correspondientes.

El método de cada signo está formado por las posiciones de las articulaciones y el tiempo que se debe esperar antes de mover otras.

Los tiempos están guardados en atributos ya que la simulación no funciona en tiempo real en todas las máquinas y algunas requieren más tiempo tanto para hacer los signos como para posicionar la mano en una posición neutral, evitando o reduciendo el riesgo de que ocurra un error por colisión y se estropee la simulación desapareciendo la mano. Esto está hecho así para que sea más fácil modificar estos tiempos en caso de que sea necesario al usarlo en otra máquina.

Capítulo 10. Conclusiones y Ampliaciones

10.1 Conclusiones

Con esta investigación se ha logrado el objetivo de demostrar la posibilidad de crear robots intérpretes y que éstos puedan ser controlados y se pueda recibir información de ellos de forma remota.

Para llegar a esta conclusión ha sido necesario estudiar la tecnología existente y "ponerla de acuerdo", y aún se puede llevar más lejos, pero lo que se pretendía demostrar ya se ha logrado.

Otra ventaja es que las aplicaciones usadas y creadas para este proyecto se pueden aprovechar en diversos modelos de robots, es decir, tienen un añadido de portabilidad. Al ser ROS un sistema basado en nodos, estos se pueden intercambiar o combinar con otros proyectos, siendo esta tecnología fácilmente adaptable a las necesidades.

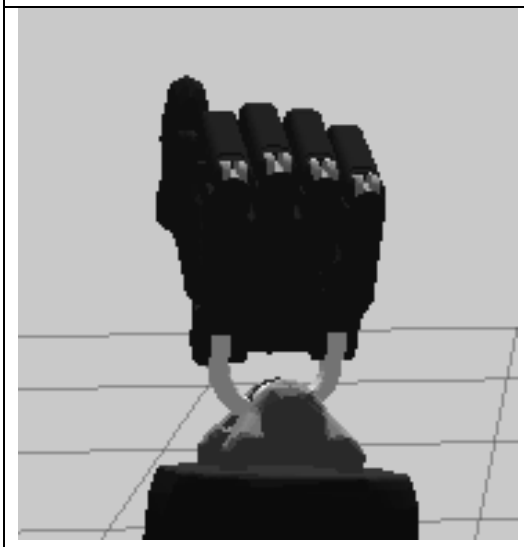
Con lo que se ha logrado se puede, por ejemplo, mantener una conversación con otra persona situada en otro lugar y que puede que hable otro idioma, usando nosotros la voz y el oído para comunicarnos mientras la otra persona, que puede padecer problemas de audición o habla, recibir los mensajes convertidos en texto y convertir sus mensajes en audio para comunicarse con nosotros.

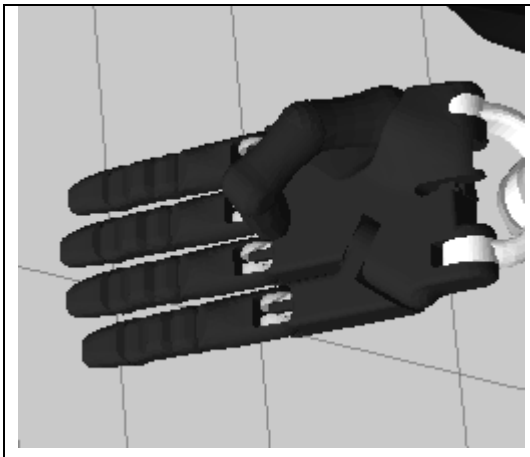
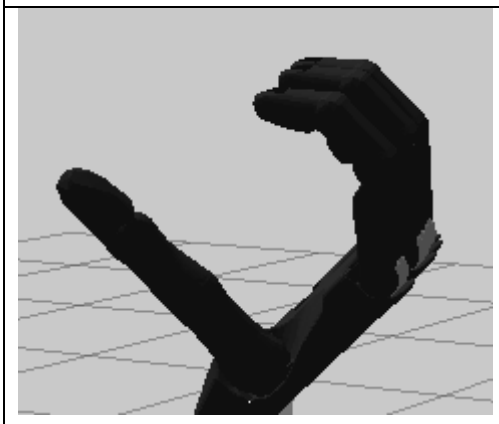
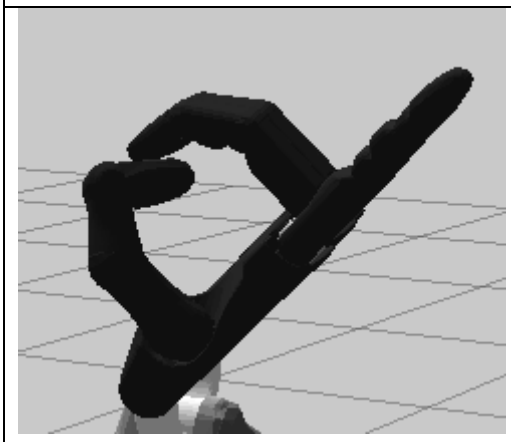
No obstante, se encuentran algunas limitaciones, el reconocimiento de voz necesita un diccionario adaptado a la situación en la que se vaya a utilizar, como una tienda donde se pueda preguntar el precio o dónde se encuentra un producto determinado. La ventaja de que en ROS los componentes estén desarrollados por módulos reside en que cuando aparezca una alternativa mejor, solo habría que cambiar esta parte y apenas modificar las demás para hacer uso de ello.

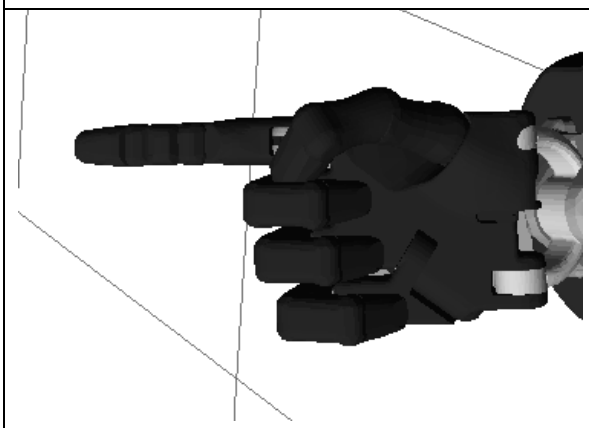
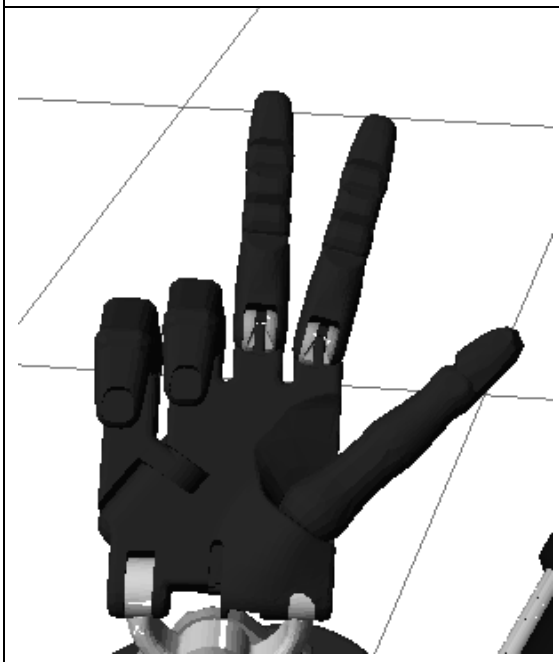
Otra limitación es la del posicionamiento de la Shadow Hand. Es una de las manos más cercanas a la mano humana, permitiendo mover los dedos y colocarlos en la posición deseada para hacer cada uno de los signos del alfabeto dactilológico. El problema reside en el movimiento de su muñeca, que es muy limitado y no permite completar la posición de la mano para representar cada signo. Esto no significa que no se pueda representar lenguaje de signos con una mano robótica, sino que significa que a la tecnología le falta poco para completar este objetivo y que en cuanto haya un interés comercial en el mismo se podrá lograr, ya que la Shadow Hand se puede acoplar a otros robots, lo cual le puede dotar de mejor movilidad y permitirle alcanzar este fin, para lo cual solo habría que añadir a lo ya hecho las instrucciones de movimiento de la muñeca y el brazo.

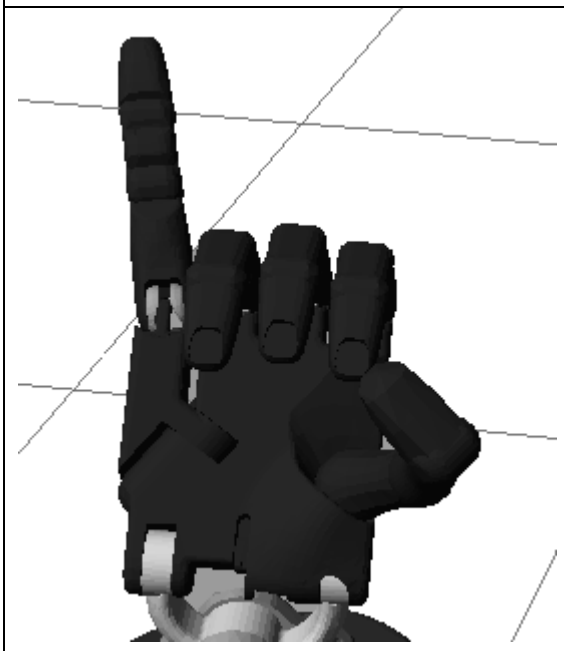
10.1.1 Representación del lenguaje de signos

Se adjunta una tabla con la representación del lenguaje de signos realizado con la versión virtual de la Shadow Hand. En esta tabla además se incluyen las mejoras o necesidades con respecto al brazo y la muñeca para completar el signo a representar. En el uso de la mano real sería ideal cubrirla con un guante o una tela para simular la piel, ya que la mano por sí sola es un esqueleto y puede dar la sensación de que el pulgar es más largo de lo normal.

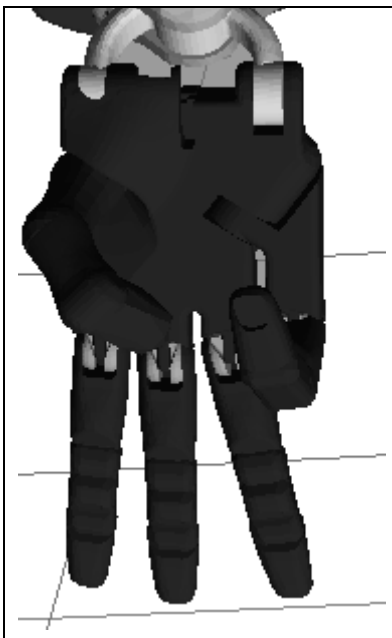
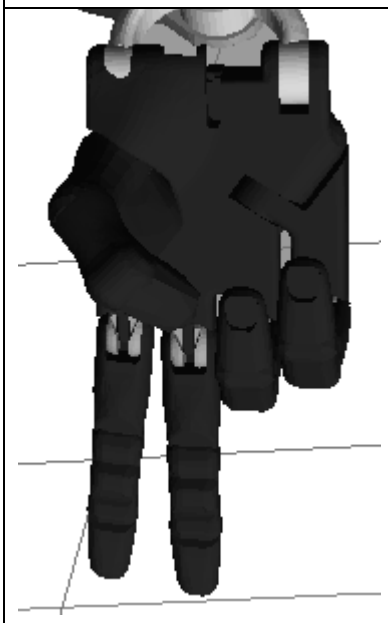
Hand	Letter	Comments
 	A	Necesario rotar el brazo 180º para conseguir esta pose.

	B	Es necesario que el brazo permita girar la muñeca 90º para alcanzar esta pose.
	C	Se puede mejorar con una ligera inclinación de muñeca de pocos grados.
	D	Se puede mejorar con una ligera inclinación de muñeca de pocos grados.
	E	Necesario rotar el brazo 180º para conseguir esta pose.

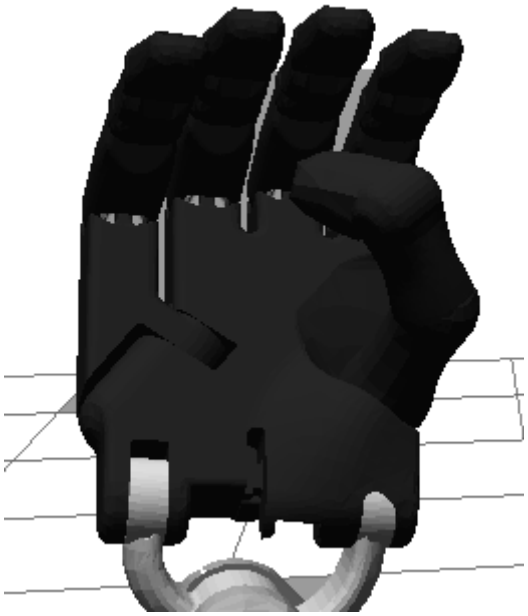
	F	Necesario rotar el brazo 90º para conseguir esta pose.
	G	Es necesario que el brazo permita girar la muñeca 90º para alcanzar esta pose y que se pueda mover delante y atrás para hacer el gesto.
	H	Falta la boca de referencia para hacer este gesto.

	I	Se necesita girar la muñeca 180º para realizar este gesto.
	J	En foto parece igual que la 'i' girada, pero en realidad el meñique está en movimiento.

	K	Se necesita un giro de 90º del brazo y una ligera inclinación de la muñeca.
	L	Requiere un giro de 180º del brazo.

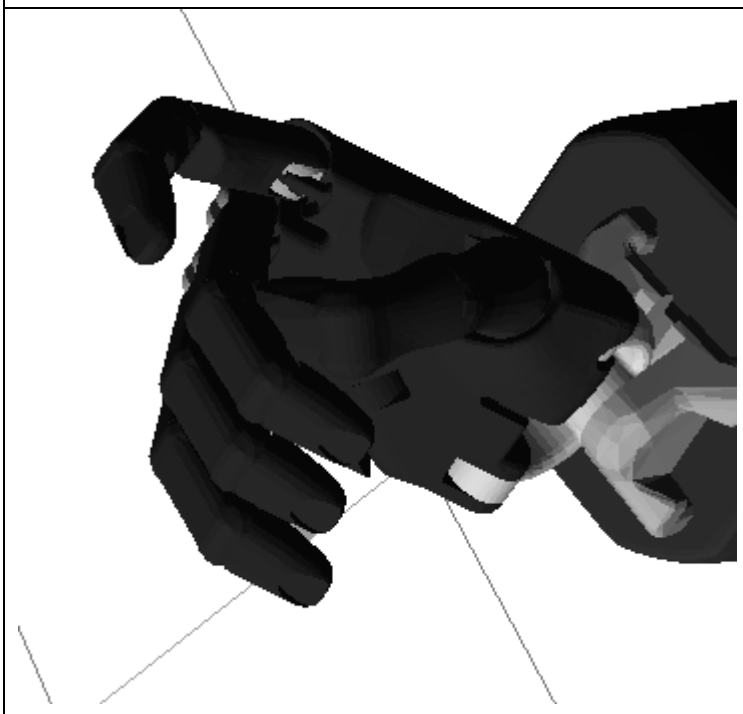
	M	Se necesita que el brazo pueda girar la mano 180º para lograr esta posición.
	N	Se necesita que el brazo pueda girar la mano 180º para lograr esta posición.

	O	Necesario rotar el brazo 90º para conseguir esta pose e inclinar ligeramente la muñeca.
	P	Necesario girar la mano 180º para ver este gesto.

	Q	
	R	

	S	Necesita un giro superior a 90º para realizar este signo.
	T	Necesita un giro superior a 90º para realizar este signo. No confundir con la S, debido a la eliminación de sombras en la simulación apenas se nota que el pulgar está por detrás del índice.

	U	Necesita un giro de 180º para representar el símbolo.
	V	Puede parecer la U en una fotografía, pero en realidad al realizar este gesto el índice y el corazón se separan y se acercan. Dada tal similitud, se muestra una imagen del acercamiento.

	W	Necesita un giro de 180º para representar el símbolo. El dedo corazón queda inmóvil, mientras que el anular y el índice se acercan y se alejan.
	X	Es necesario girar la muñeca 90º. El dedo índice se mueve ligeramente hacia los lados.

 	Y	Necesita un giro de 180º para representar el símbolo. El meñique permanece en movimiento durante la representación del símbolo, plegándose y estirándose.

	Z	Similar al anterior, pero el meñique se mantiene estirado y se mueve de forma zigzagueante.

10.2 Ampliaciones

Las ampliaciones posibles se resumen en añadir opciones alternativas que se puedan combinar con el resto de partes del proyecto, como pueden ser:

- Usar otro sistema de síntesis de voz.
- Usar otra mano robótica o virtual para la realización de signos, desarrollando su correspondiente nodo de control.
- Cambiar el sistema de traducción de la página web por uno mejor que se pueda utilizar o se contrate (ver el apartado 9.4.2).
- Ampliar o mejorar el diccionario del reconocimiento de voz o cambiar el sistema de reconocimiento de voz, lo cual significaría modificar los nodos de control correspondientes.

Otra ampliación importante sería la de mejorar el control de la Shadow Hand si, como se ha mencionado antes, se adaptase para tener una mejor movilidad y una mejor muñeca al incluirla en otro robot, ya que según la página web de la desarrolladora la mano puede ser adaptada para otros robots y otras necesidades[11].

[11] Shadow Dexterous Hand™: http://www.shadowrobot.com/products/dexterous-hand/

Capítulo 11. Presupuesto

Concepto	Cantidad / Meses de trabajo	Precio Unitario	Coste Total Concepto
Preparación del material	0,12	1.500,00 €	180,00 €
Estudio del estado del arte	2,50	1.500,00 €	3.750,00 €
Desarrollo del proyecto	2,00	1.500,00 €	3.000,00 €
Documentación	1,00	1.500,00 €	1.500,00 €
Material para el desarrollo	1,00	600,00 €	600,00 €
		Subtotal	9.030,00 €
		IVA (21%)	1.896,30 €
		TOTAL	**10.926,30 €**

Capítulo 12. Referencias Bibliográficas

12.1 Libros y Artículos

[1] LANFRANCO, Anthony R., et al. Robotic surgery: a current perspective. Annals of Surgery, 2004, vol. 239, no 1, p. 14.

[2] MARESCAUX, Jacques, et al. Transcontinental robot-assisted remote telesurgery: feasibility and potential applications. Annals of surgery, 2002, vol. 235, no 4, p. 487.

[3] YAMAMOTO, Shun'ichi, et al. Enhanced robot speech recognition based on microphone array source separation and missing feature theory. En Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on. IEEE, 2005. p. 1477-1482.

[4] GOCKLEY, Rachel, et al. Designing robots for long-term social interaction. En Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on. IEEE, 2005. p. 1338-1343.

[5] KOZIMA, Hideki; NAKAGAWA, Cocoro; YASUDA, Yuriko. Interactive robots for communication-care: A case-study in autism therapy. En Robot and Human Interactive Communication, 2005. ROMAN 2005. IEEE International Workshop on. IEEE, 2005. p. 341-346.

[6] LEE, Jun Ki, et al. The design of a semi-autonomous robot avatar for family communication and education. En Robot and Human Interactive Communication, 2008. RO-MAN 2008. The 17th IEEE International Symposium on. IEEE, 2008. p. 166-173.

[7] TELLEZ, Ricardo, et al. Reem-B: An autonomous lightweight human-size humanoid robot. En Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on. IEEE, 2008. p. 462-468.

[8] PACK, R. T.; WILKES, D. Mitchell; KAWAMURA, Kazuhiko. A software architecture for integrated service robot development. En Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on. IEEE, 1997. p. 3774-3779.

[9] BRUYNINCKX, Herman. Open robot control software: the OROCOS project. En Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on. IEEE, 2001. p. 2523-2528.

[10] QUIGLEY, Morgan, et al. ROS: an open-source Robot Operating System. En ICRA workshop on open source software. 2009. p. 5.

[11] MCDONALD, John, et al. An improved articulated model of the human hand. The Visual Computer, 2001, vol. 17, no 3, p. 158-166.

[12] Gilbertson, R. G. *Muscle Wires Project Book*. California: Mondo-tronics, Inc., 1994, Third Ed.

[13] SIMMONS, Reid, et al. 5 Xavier: An Autonomous Mobile Robot on the Web. Beyond Webcams: an introduction to online robots, 2002, p. 81.

[14] THRUN, Sebastian, et al. MINERVA: A second-generation museum tour-guide robot. En Robotics and automation, 1999. Proceedings. 1999 IEEE international conference on. IEEE, 1999.

[15] ROTHLING, Frank, et al. Platform portable anthropomorphic grasping with the bielefeld 20-dof shadow and 9-dof tum hand. En Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on. IEEE, 2007. p. 2951-2956.

[16] HUGGINS-DAINES, David, et al. Pocketsphinx: A free, real-time continuous speech recognition system for hand-held devices. En Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on. IEEE, 2006. p. I-I.

[17] VERTANEN, Keith. Baseline WSJ acoustic models for HTK and Sphinx: Training recipes and recognition experiments. Technical report). Cambridge, United Kingdom: Cavendish Laboratory, 2006.

[18] BLACK, Alan W., et al. The festival speech synthesis system. University of Edinburgh, 2002, vol. 1.

12.2 Referencias en Internet

- **ROS:** http://www.ros.org/
- **ROS Wiki:** http://wiki.ros.org/
    - Distintas secciones son referenciadas a lo largo del documento acorde con la relación del apartado en el que están incluidas.
- **ROS by Example: Speech Recognition and Text-to-Speech (TTS) [Patrick Goebel]:** http://www.pirobot.org/blog/0022/
- **Órdenes y confirmación mediante voz (sphinx+festival) - Robótica - Universidad de León**: http://robotica.unileon.es/mediawiki/index.php/Órdenes_y_confirmación_mediante_voz_(sphinx%2Bfestival)
- **Sphinx Knowledge Base Tool:** http://www.speech.cs.cmu.edu/tools/lmtool-new.html
- **Pocketsphinx - Adding words and Improving accuracy:** http://stackoverflow.com/questions/4535208/pocketsphinx-adding-words-and-improving-accuracy
- **Sound_play Documentation:** http://docs.ros.org/api/sound_play/html/index.html
- **Frengly.com:** http://www.frengly.com/
- **XML DOM Load Functions:** http://www.w3schools.com/dom/dom_loadxmldoc.asp
- **Introduction to Robotics [Mr. Roi Yehoshua]:** http://u.cs.biu.ac.il/~yehoshr1/89-685/

Capítulo 13. Apéndices

13.1 Glosario y Diccionario de Datos

- **Control remoto:** Dispositivo o programa que regula a distancia el funcionamiento de un aparato, mecanismo o sistema.
- **Intérprete:** Persona o mecanismo que explica a otras, en lengua que entienden, lo dicho en otra que les es desconocida.
- **Lengua de signos:** Sistema de comunicación mediante el movimiento y el posicionamiento de las manos, los dedos y la boca.
- **Reconocimiento de voz:** Aplicación informática que analiza un archivo de audio para sintetizar texto a partir del mismo.
- **Robótica:** Técnica que aplica la informática al diseño y empleo de aparatos que, en sustitución de personas, realizan operaciones o trabajos.
- **Simulación:** Representación de algo, que puede o no ser real.
- **Sintetizador de voz:** Aplicación informática que permite la simulación de una voz humana para la lectura de textos.

13.2 Contenido Entregado en el CD-ROM

13.2.1 Contenidos

13.2.1.1.1 Directorio raíz del disco

- **Documentacion.pdf:** Presente archivo con la documentación del proyecto.
- **Paper.pdf:** Paper enviado a *IEEE Robotics & Automation Magazine*.
- **Review.pdf:** Review enviado a *Autonomous Robots*.
- **Proyecto:** Directorio en el que se encuentra el proyecto.
- **Web:** Incluye los archivos de las páginas web creadas en el proyecto para la comunicación con el robot.
- **Autorun.inf:** Carga el icono del disco.
- **Icono.ico:** Icono del disco.

13.2.1.1.2 Directorio proyecto

- **sandbox:** Directorio en el que se encuentran las partes del proyecto. En este caso solo incluye el directorio **interprete**.

Los siguientes archivos fueron generados automáticamente durante la creación del paquete y del área de trabajo de ROS y son necesarios para poder invocar los recursos del proyecto:

- **.rosinstall**
- **.rosinstall.bak**
- **setup.bash**
- **setup.sh**
- **setup.zsh**

13.2.1.1.3 Directorio interprete

- **build:** Directorio en el que se almacenan los archivos de la creación de paquetes y de caché.
- **config:** En este directorio solo se encuentran los archivos correspondientes al diccionario y al modelo de lenguaje que se usará para el reconocimiento de voz.

- **launch:** Incluye el archivo **comandos.launch** necesario para usar el diccionario que hemos creado para el proyecto.
- **scripts:** Incluye los archivos "**.py**" de los nodos que se han creado para este proyecto.

Los siguientes archivos fueron generados automáticamente durante la creación del paquete y del área de trabajo de ROS y son necesarios para poder invocar los recursos del proyecto:

- **CMakeLists.txt**
- **mainpage.dox**
- **Makefile**
- **manifest.xml**

13.2.2 Código Ejecutable

Los archivos ejecutables se encuentran en el directorio script y son los siguientes:

- **controlSH.py:** Permite controlar la mano robótica desde la página web.
- **controlSHVoz.py:** Permite controlar la mano robótica por voz.
- **escuchaWeb.py:** Envía los mensajes recibidos de la página web al sintetizador de voz para que los diga en inglés.
- **escuchaWebEsp.py:** Envía los mensajes recibidos de la página web al sintetizador de voz para que los diga en castellano.

13.2.2.1 Páginas web

La página web usada en el proyecto, que se encuentran en el directorio Web, incluye dos versiones:

- **saluda.html:** Se comunica con el robot sin alterar los mensajes.
- **saluda_traduce.html:** Se comunica con el robot traduciendo del castellano al inglés los mensajes que envía y traduciendo del inglés al castellano los mensajes recibidos.

13.2.3 Ficheros de Configuración

Para usar el reconocimiento de voz se necesitan los archivos del diccionario y del modelo de lenguaje, los cuales se encuentran en el directorio config dentro del directorio intérprete (ver apartado 13.2.1.1.3), además del lanzador comandos.launch incluido en el directorio launch.

13.3 Índice Alfabético

C

Comandos de voz, 9

H

Habla a texto, 9

I

índice alfabético, 177
Intérprete, 9, 171

L

Lengua de signos, 9, 171
Lenguaje de signos, 9, 30, 40

M

Mano robótica, 9, 42, 71, 74, 82

P

problemas encontrados, 119
pruebas unitarias, 78, 101, 102, 130, 175

R

Reconocimiento de voz, 9, 26, 39, 40, 76, 101, 119, 129, 171
Redondo L., J. Manuel, 169
Remoto, 9
Robot, 7, 9, 30, 31, 53, 59, 61, 63, 64, 82, 85, 115, 127, 167
Robot Operative System, 9
ROS, 7, 9, 31, 32, 40, 47, 48, 53, 55, 74, 85, 100, 102, 115, 116, 118, 127, 151, 152, 153, 156, 161, 169, 172, 173

T

Texto a habla, 9

W

Web, 9, 31, 111, 139, 140, 141, 142, 143, 169, 170, 172, 174, 175, 176

13.4 Código Fuente

13.4.1 Nodos para el habla del robot

13.4.1.1 escuchaWeb.py

```
#!/usr/bin/env python
# license removed for brevity
import roslib; roslib.load_manifest('sound_play')
import rospy
from std_msgs.msg import String
from sound_play.msg import SoundRequest

from sound_play.libsoundplay import SoundClient

def sleep(t):
    try:
        rospy.sleep(t) #duerme el programa, para cuando haya que
esperar
    except:
        pass

def callback(data):
    # Ha recibido el mensaje
    rospy.loginfo(rospy.get_caller_id()+" I heard %s",data.data)
    soundhandle = SoundClient()
    # Espera un segundo a que se prepare
    rospy.sleep(1)
    # Inicia el habla
    soundhandle.say(data.data)
    # Deja un margen para que el robot hable
    rospy.sleep(5)

def listener():

    # Iniciamos el nodo de escucha
    rospy.init_node('listener', anonymous=True)

    # Se suscribe al canal chatter
    rospy.Subscriber("chatter", String, callback)

    # spin() evita que se detenga, crea un ciclo como un while(true)
en Java
    rospy.spin()

if __name__ == '__main__':
    listener()
```

13.4.1.2 escuchaWebEsp.py

```
#!/usr/bin/env python
# license removed for brevity
```

```
import roslib; roslib.load_manifest('sound_play')
import rospy
from std_msgs.msg import String
from sound_play.msg import SoundRequest

from sound_play.libsoundplay import SoundClient

def sleep(t):
    try:
        rospy.sleep(t) #duerme el programa, para cuando haya que
esperar
    except:
        pass

def callback(data):
    # Ha recibido el mensaje
    rospy.loginfo(rospy.get_caller_id()+" Recibido: %s",data.data)
#escribe lo que escucho
    # Se debe tener instalada la voz indicada para que hable en
Castellano, se puede cambiar por otra.
    # sudo apt-get install festvox-ellpc11k
    # Comprobamos que esta instalada con: ls
/usr/share/festival/voices/spanish
    # Si en los resultados aparece el_diphone, esta correcto.
    voice = rospy.get_param("~voice", "voice_el_diphone")
    soundhandle = SoundClient()
    # Espera un segundo a que se prepare
    rospy.sleep(1)
    # Inicia el habla con la voz indicada
    soundhandle.say(data.data,voice)
    # Deja un margen para que el robot hable
    rospy.sleep(5)

def listener():

    # Iniciamos el nodo de escucha
    rospy.init_node('listener', anonymous=True)

    # Se suscribe al canal chatter
    rospy.Subscriber("chatter", String, callback)

    # spin() evita que se detenga, crea un ciclo como un while(true)
en Java
    rospy.spin()

if __name__ == '__main__':
    listener()
```

13.4.2 Nodos para el control de la Shadow Hand

13.4.2.1 controlSH.py

```
#!/usr/bin/env python

import roslib; roslib.load_manifest('sr_example')
import rospy
import time
from std_msgs.msg import Float64, String
from pr2_controllers_msgs.msg import JointControllerState
```

```
# type of controller that is running
controller_type = "_position_controller"

#pub = rospy.Publisher('sh_' + child_name + controller_type +
'/command', Float64)
pubSass = rospy.Publisher('sa_ss' + controller_type + '/command',
Float64)
pubFfj3 = rospy.Publisher('sh_ffj3' + controller_type + '/command',
Float64)
pubFfj0 = rospy.Publisher('sh_ffj0' + controller_type + '/command',
Float64)
pubFfj4 = rospy.Publisher('sh_ffj4' + controller_type + '/command',
Float64)
pubMfj4 = rospy.Publisher('sh_mfj4' + controller_type + '/command',
Float64)
pubMfj3 = rospy.Publisher('sh_mfj3' + controller_type + '/command',
Float64)
pubMfj0 = rospy.Publisher('sh_mfj0' + controller_type + '/command',
Float64)
pubRfj4 = rospy.Publisher('sh_rfj4' + controller_type + '/command',
Float64)
pubRfj3 = rospy.Publisher('sh_rfj3' + controller_type + '/command',
Float64)
pubRfj0 = rospy.Publisher('sh_rfj0' + controller_type + '/command',
Float64)
pubLfj5 = rospy.Publisher('sh_lfj5' + controller_type + '/command',
Float64)
pubLfj4 = rospy.Publisher('sh_lfj4' + controller_type + '/command',
Float64)
pubLfj3 = rospy.Publisher('sh_lfj3' + controller_type + '/command',
Float64)
pubLfj0 = rospy.Publisher('sh_lfj0' + controller_type + '/command',
Float64)
pubThj5 = rospy.Publisher('sh_thj5' + controller_type + '/command',
Float64)
pubThj4 = rospy.Publisher('sh_thj4' + controller_type + '/command',
Float64)
pubThj3 = rospy.Publisher('sh_thj3' + controller_type + '/command',
Float64)
pubThj2 = rospy.Publisher('sh_thj2' + controller_type + '/command',
Float64)
pubThj1 = rospy.Publisher('sh_thj1' + controller_type + '/command',
Float64)
tiempo_mucho = 9
tiempo_medio = 5
tiempo_poco = 3
tiempo_poquisimo = 1

def cierra():
    pubThj5.publish(1.5)
    pubThj4.publish(1.5)
    pubThj3.publish(1.5)
    pubThj2.publish(1.0)
    time.sleep(tiempo_poco)
    pubFfj3.publish(1.5)
    pubFfj0.publish(2.5)
    #pubFfj4.publish(1.5)
    pubMfj3.publish(1.5)
    pubMfj0.publish(2.5)
    pubRfj3.publish(1.5)
    pubRfj0.publish(2.5)
    pubLfj3.publish(1.5)
```

```
    pubLfj0.publish(2.5)
    #pubSass.publish(1.5)
    time.sleep(tiempo_medio)

def abre():
    pubFfj3.publish(0.0)
    pubFfj0.publish(0.0)
    pubFfj4.publish(0.0)
    pubMfj3.publish(0.0)
    pubMfj0.publish(0.0)
    pubMfj4.publish(0.0)
    pubRfj3.publish(0.0)
    pubRfj0.publish(0.0)
    pubRfj4.publish(0.0)
    pubLfj3.publish(0.0)
    pubLfj0.publish(0.0)
    pubLfj4.publish(0.0)
    time.sleep(tiempo_poco)
    pubThj5.publish(0.0)
    pubThj4.publish(0.0)
    pubThj3.publish(0.0)
    pubThj2.publish(0.0)
    pubThj1.publish(0.0)
    #pubSass.publish(1.75)
    time.sleep(tiempo_medio)

def abre2():
    pubThj5.publish(0.0)
    pubThj4.publish(0.0)
    pubThj3.publish(0.0)
    pubThj2.publish(0.0)
    pubThj1.publish(0.0)
    time.sleep(tiempo_poco)
    pubFfj3.publish(0.0)
    pubFfj0.publish(0.0)
    pubFfj4.publish(0.0)
    pubMfj3.publish(0.0)
    pubMfj0.publish(0.0)
    pubMfj4.publish(0.0)
    pubRfj3.publish(0.0)
    pubRfj0.publish(0.0)
    pubRfj4.publish(0.0)
    pubLfj3.publish(0.0)
    pubLfj0.publish(0.0)
    pubLfj4.publish(0.0)
    #pubSass.publish(1.75)
    time.sleep(tiempo_medio)

def abre3():
    pubThj5.publish(0.0)
    pubThj4.publish(0.0)
    pubThj3.publish(0.0)
    pubThj2.publish(0.0)
    pubThj1.publish(0.0)
    pubFfj3.publish(0.0)
    pubFfj0.publish(0.0)
    pubFfj4.publish(0.0)
    pubMfj3.publish(0.0)
    pubMfj0.publish(0.0)
    pubMfj4.publish(0.0)
    pubRfj3.publish(0.0)
    pubRfj0.publish(0.0)
    pubRfj4.publish(0.0)
```

```
    pubLfj3.publish(0.0)
    pubLfj0.publish(0.0)
    pubLfj4.publish(0.0)
    #pubSass.publish(1.75)
    time.sleep(tiempo_poco)

def a():
    pubFfj3.publish(1.5)
    pubFfj0.publish(2.5)
    #pubFfj4.publish(1.5)
    pubMfj3.publish(1.5)
    pubMfj0.publish(2.5)
    pubRfj3.publish(1.5)
    pubRfj0.publish(2.5)
    pubLfj3.publish(1.5)
    pubLfj0.publish(2.5)
    #pubSass.publish(1.5)
    time.sleep(tiempo_medio)
    pubThj5.publish(0.5)
    pubThj4.publish(1.5)
    pubThj3.publish(1.0)
    pubThj2.publish(0.5)
    #pubThj1.publish(1.5)
    time.sleep(tiempo_mucho)

def b():
    pubThj5.publish(0.6)
    pubThj4.publish(1.5)
    pubThj3.publish(1.5)
    pubThj2.publish(1.5)
    pubThj1.publish(1.5)
    time.sleep(tiempo_mucho)

def c():
    pubThj4.publish(1.5)
    pubThj3.publish(1.5)
    pubFfj3.publish(0.8)
    pubFfj0.publish(1.5)
    pubMfj3.publish(0.8)
    pubMfj0.publish(1.5)
    pubRfj3.publish(0.8)
    pubRfj0.publish(1.5)
    pubLfj3.publish(0.8)
    pubLfj0.publish(1.5)
    time.sleep(tiempo_mucho)

def d():
    pubThj4.publish(1.5)
    pubThj3.publish(1.5)
    pubThj2.publish(1.5)
    pubThj1.publish(1.5)
    pubMfj3.publish(1.5)
    pubMfj0.publish(1.5)
    pubRfj3.publish(1.5)
    pubRfj0.publish(1.5)
    pubLfj3.publish(1.5)
    pubLfj0.publish(1.5)
    time.sleep(tiempo_mucho)

def e():
    pubFfj3.publish(1.5)
    pubFfj0.publish(1.5)
    #pubFfj4.publish(1.5)
```

```
    pubMfj3.publish(1.5)
    pubMfj0.publish(1.5)
    pubRfj3.publish(1.5)
    pubRfj0.publish(1.5)
    pubLfj3.publish(1.5)
    pubLfj0.publish(1.5)
    #pubSass.publish(1.5)
    time.sleep(tiempo_mucho)

def f():
    #pubThj5.publish(0.5)
    #pubThj4.publish(1.0)
    pubThj3.publish(1.5)
    pubThj2.publish(1.0)
    pubFfj3.publish(1.5)
    pubMfj3.publish(0.6)
    pubRfj3.publish(0.3)
    time.sleep(tiempo_mucho)

def g():
    #pubFfj3.publish(1.5)
    #pubFfj0.publish(2.5)
    #pubFfj4.publish(1.5)
    pubMfj3.publish(1.5)
    pubMfj0.publish(2.5)
    pubRfj3.publish(1.5)
    pubRfj0.publish(2.5)
    pubLfj3.publish(1.5)
    pubLfj0.publish(2.5)
    #pubSass.publish(1.5)
    time.sleep(tiempo_medio)
    pubThj5.publish(0.4)
    pubThj4.publish(1.0)
    pubThj3.publish(0.2)
    pubThj2.publish(0.4)
    pubThj1.publish(1.5)
    time.sleep(tiempo_mucho)

def h():
    #pubFfj3.publish(1.5)
    #pubFfj0.publish(1.5)
    pubFfj4.publish(-0.2)
    #pubMfj3.publish(1.5)
    #pubMfj0.publish(1.5)
    #pubMfj4.publish(-0.1)
    pubRfj3.publish(1.5)
    pubRfj0.publish(1.5)
    pubLfj3.publish(1.5)
    pubLfj0.publish(1.5)
    time.sleep(tiempo_mucho)

def i():
    pubFfj3.publish(1.5)
    pubFfj0.publish(1.5)
    #pubFfj4.publish(1.5)
    pubMfj3.publish(1.5)
    pubMfj0.publish(1.5)
    pubRfj3.publish(1.5)
    pubRfj0.publish(1.5)
    #pubLfj3.publish(1.5)
    #pubLfj0.publish(2.5)
    #pubSass.publish(1.5)
    time.sleep(tiempo_poco)
```

```
    pubThj5.publish(-0.5)
    pubThj4.publish(1.2)
    pubThj3.publish(0.2)
    pubThj2.publish(0.4)
    pubThj1.publish(1.5)
    time.sleep(tiempo_mucho)

def j():
    pubFfj3.publish(1.5)
    pubFfj0.publish(1.5)
    #pubFfj4.publish(1.5)
    pubMfj3.publish(1.5)
    pubMfj0.publish(1.5)
    pubRfj3.publish(1.5)
    pubRfj0.publish(1.5)
    #pubLfj3.publish(1.5)
    #pubLfj0.publish(2.5)
    #pubSass.publish(1.5)
    pubLfj4.publish(0.2)
    time.sleep(tiempo_poco)
    pubThj5.publish(-0.5)
    pubThj4.publish(1.2)
    pubThj3.publish(0.2)
    pubThj2.publish(0.4)
    pubThj1.publish(1.5)
    pubLfj4.publish(-0.2)
    time.sleep(tiempo_poco)
    pubLfj4.publish(0.2)
    time.sleep(tiempo_poco)
    pubLfj4.publish(-0.2)
    time.sleep(tiempo_poco)
    pubLfj4.publish(0.0)

def k():
    pubThj5.publish(-0.5)
    pubThj4.publish(1.2)
    pubMfj3.publish(1.5)
    pubMfj0.publish(1.5)
    pubRfj3.publish(1.5)
    pubRfj0.publish(1.5)
    pubLfj3.publish(1.5)
    pubLfj0.publish(1.5)
    pubFfj3.publish(1.0)
    time.sleep(tiempo_mucho)

def l():
    pubThj5.publish(-0.9)
    pubMfj3.publish(1.5)
    pubMfj0.publish(1.5)
    pubRfj3.publish(1.5)
    pubRfj0.publish(1.5)
    pubLfj3.publish(1.5)
    pubLfj0.publish(1.5)
    time.sleep(tiempo_mucho)

def m():
    pubThj4.publish(1.5)
    pubThj3.publish(1.5)
    pubThj2.publish(1.5)
    pubThj1.publish(1.5)
    pubLfj3.publish(1.5)
    pubLfj0.publish(1.5)
    pubLfj4.publish(-0.2)
```

```
    pubRfj4.publish(-0.2)
    pubFfj4.publish(-0.1)
    time.sleep(tiempo_mucho)

def n():
    pubThj4.publish(1.5)
    pubThj3.publish(1.5)
    pubThj2.publish(1.5)
    pubThj1.publish(1.5)
    pubLfj3.publish(1.5)
    pubLfj0.publish(1.5)
    pubRfj3.publish(1.5)
    pubRfj0.publish(1.5)
    pubFfj4.publish(-0.1)
    time.sleep(tiempo_mucho)

def nn():
    pubThj4.publish(1.5)
    pubThj3.publish(1.5)
    pubThj2.publish(1.5)
    pubThj1.publish(1.5)
    pubLfj3.publish(1.5)
    pubLfj0.publish(1.5)
    pubRfj3.publish(1.5)
    pubRfj0.publish(1.5)
    pubFfj4.publish(-0.1)
    pubMfj4.publish(0.1)
    time.sleep(tiempo_poco)
    pubFfj4.publish(0.0)
    pubMfj4.publish(0.0)
    time.sleep(tiempo_poco)
    pubFfj4.publish(-0.1)
    pubMfj4.publish(0.1)
    time.sleep(tiempo_poco)
    pubFfj4.publish(0.0)
    pubMfj4.publish(0.0)
    time.sleep(tiempo_medio)

def o():
    time.sleep(tiempo_poquisimo)
    pubThj4.publish(1.4)
    pubThj3.publish(1.4)
    pubThj2.publish(1.4)
    pubThj1.publish(1.4)
    pubFfj3.publish(1.4)
    pubFfj0.publish(1.4)
    pubMfj3.publish(0.7)
    pubRfj3.publish(0.4)
    time.sleep(tiempo_mucho)

def p():
    pubThj4.publish(1.5)
    pubThj3.publish(1.5)
    pubThj2.publish(1.5)
    pubThj1.publish(1.5)
    pubLfj3.publish(1.5)
    pubLfj0.publish(1.5)
    time.sleep(tiempo_mucho)

def q():
    pubThj4.publish(1.5)
    pubThj3.publish(1.5)
    pubThj2.publish(1.5)
```

```
    pubThj1.publish(1.5)
    pubFfj3.publish(1.5)
    pubFfj0.publish(0.5)
    pubMfj3.publish(1.5)
    pubMfj0.publish(0.5)
    pubRfj3.publish(1.5)
    pubRfj0.publish(0.5)
    pubLfj3.publish(1.5)
    pubLfj0.publish(0.5)
    time.sleep(tiempo_mucho)

def r():
    pubThj4.publish(1.5)
    pubThj3.publish(1.5)
    pubThj2.publish(1.5)
    pubThj1.publish(1.5)
    pubLfj3.publish(1.5)
    pubLfj0.publish(1.5)
    pubRfj3.publish(1.5)
    pubRfj0.publish(1.5)
    pubFfj3.publish(0.5)
    pubFfj4.publish(0.2)
    pubMfj4.publish(-0.2)
    time.sleep(tiempo_mucho)

def o():
    pubThj4.publish(1.0)
    pubThj3.publish(0.8)
    pubThj2.publish(0.0)
    pubThj1.publish(0.0)
    pubFfj3.publish(1.5)
    pubFfj0.publish(1.5)
    pubMfj3.publish(0.7)
    pubRfj3.publish(0.4)
    time.sleep(tiempo_mucho)

def s():
    pubThj5.publish(0.95)
    pubThj4.publish(0.0)
    pubThj3.publish(0.0)
    pubThj2.publish(0.0)
    pubThj1.publish(0.0)
    time.sleep(tiempo_poco)
    pubFfj3.publish(1.5)
    pubFfj0.publish(1.5)
    pubMfj3.publish(0.7)
    pubRfj3.publish(0.4)
    time.sleep(tiempo_mucho)

def t():
    pubThj5.publish(0.95)
    pubThj4.publish(1.2)
    pubThj3.publish(0.2)
    pubThj2.publish(0.43)
    pubThj1.publish(0.0)
    time.sleep(tiempo_poco)
    pubFfj3.publish(1.5)
    pubFfj0.publish(1.5)
    pubMfj3.publish(0.7)
    pubRfj3.publish(0.4)
    time.sleep(tiempo_mucho)

def u():
```

```
    pubThj4.publish(1.5)
    pubThj3.publish(1.5)
    pubThj2.publish(1.5)
    pubThj1.publish(1.5)
    pubLfj3.publish(1.5)
    pubLfj0.publish(1.5)
    pubRfj3.publish(1.5)
    pubRfj0.publish(1.5)
    pubFfj4.publish(-0.1)
    time.sleep(tiempo_mucho)

def v():
    pubThj4.publish(1.5)
    pubThj3.publish(1.5)
    pubThj2.publish(1.5)
    pubThj1.publish(1.5)
    pubLfj3.publish(1.5)
    pubLfj0.publish(1.5)
    pubRfj3.publish(1.5)
    pubRfj0.publish(1.5)
    pubFfj4.publish(-0.1)
    pubMfj4.publish(0.1)
    time.sleep(tiempo_poco)
    pubFfj4.publish(0.0)
    pubMfj4.publish(0.0)
    time.sleep(tiempo_poco)
    pubFfj4.publish(-0.1)
    pubMfj4.publish(0.1)
    time.sleep(tiempo_poco)
    pubFfj4.publish(0.0)
    pubMfj4.publish(0.0)
    time.sleep(tiempo_medio)

def w():
    pubThj4.publish(1.5)
    pubThj3.publish(1.5)
    pubThj2.publish(1.5)
    pubThj1.publish(1.5)
    pubLfj3.publish(1.5)
    pubLfj0.publish(1.5)
    pubLfj4.publish(-0.2)
    pubRfj4.publish(-0.2)
    pubFfj4.publish(-0.1)
    time.sleep(tiempo_poco)
    pubFfj4.publish(0.0)
    pubRfj4.publish(0.0)
    time.sleep(tiempo_poco)
    pubFfj4.publish(-0.1)
    pubRfj4.publish(-0.2)
    time.sleep(tiempo_poco)
    pubFfj4.publish(0.0)
    pubRfj4.publish(0.0)
    time.sleep(tiempo_medio)

def x():
    pubFfj3.publish(0.5)
    pubFfj0.publish(1.5)
    pubMfj3.publish(1.5)
    pubMfj0.publish(1.5)
    pubRfj3.publish(1.5)
    pubRfj0.publish(1.5)
    pubLfj3.publish(1.5)
    pubLfj0.publish(1.5)
```

```
    pubFfj4.publish(-0.2)
    #pubSass.publish(1.5)
    time.sleep(tiempo_poco)
    pubFfj4.publish(0.2)
    pubThj5.publish(0.4)
    pubThj4.publish(1.0)
    pubThj3.publish(0.2)
    pubThj2.publish(0.4)
    pubThj1.publish(1.5)
    time.sleep(tiempo_poco)
    pubFfj4.publish(-0.2)
    time.sleep(tiempo_poco)
    pubFfj4.publish(0.2)
    time.sleep(tiempo_poco)

def y():
    pubFfj3.publish(1.5)
    pubFfj0.publish(1.5)
    #pubFfj4.publish(1.5)
    pubMfj3.publish(1.5)
    pubMfj0.publish(1.5)
    pubRfj3.publish(1.5)
    pubRfj0.publish(1.5)
    #pubLfj3.publish(1.5)
    #pubLfj0.publish(2.5)
    #pubSass.publish(1.5)
    pubLfj3.publish(0.5)
    pubLfj0.publish(1.5)
    time.sleep(tiempo_poco)
    pubLfj0.publish(0.0)
    pubThj5.publish(-0.5)
    pubThj4.publish(1.2)
    pubThj3.publish(0.2)
    pubThj2.publish(0.4)
    pubThj1.publish(1.5)
    time.sleep(tiempo_poco)
    pubLfj0.publish(1.5)
    time.sleep(tiempo_poco)
    pubLfj0.publish(0.0)
    time.sleep(tiempo_medio)

def z():
    pubFfj3.publish(1.5)
    pubFfj0.publish(1.5)
    #pubFfj4.publish(1.5)
    pubMfj3.publish(1.5)
    pubMfj0.publish(1.5)
    pubRfj3.publish(1.5)
    pubRfj0.publish(1.5)
    #pubLfj3.publish(1.5)
    #pubLfj0.publish(2.5)
    #pubSass.publish(1.5)
    pubLfj4.publish(-0.2)
    time.sleep(tiempo_poco)
    pubLfj4.publish(0.2)
    pubThj5.publish(-0.5)
    pubThj4.publish(1.2)
    pubThj3.publish(0.2)
    pubThj2.publish(0.4)
    pubThj1.publish(1.5)
    time.sleep(tiempo_poco)
    pubLfj4.publish(-0.2)
    pubLfj3.publish(0.5)
```

```
    time.sleep(tiempo_poco)
    pubLfj4.publish(0.2)
    time.sleep(tiempo_medio)

def mover(data):
    abre3()
    # Ha recibido el mensaje
    rospy.loginfo(rospy.get_caller_id()+" Recibido: %s",data.data)
#escribe lo que escucho
    cadena = data.data
    print cadena
    lista = list(cadena)
    for x in lista:
        print x
        if x == 'a':
            a()
        elif x == 'A':
            a()
        elif x == 'b':
            b()
        elif x == 'B':
            b()
        elif x == 'c':
            c()
        elif x == 'C':
            c()
        elif x == 'd':
            d()
        elif x == 'D':
            d()
        elif x == 'e':
            e()
        elif x == 'E':
            e()
        elif x == 'f':
            f()
        elif x == 'F':
            f()
        elif x == 'g':
            g()
        elif x == 'G':
            g()
        elif x == 'h':
            h()
        elif x == 'H':
            h()
        elif x == 'i':
            i()
        elif x == 'I':
            i()
        elif x == 'j':
            j()
        elif x == 'J':
            j()
        elif x == 'k':
            k()
        elif x == 'K':
            k()
        elif x == 'l':
            l()
        elif x == 'L':
            l()
        elif x == 'm':
```

```
            m()
        elif x == 'M':
            m()
        elif x == 'n':
            n()
        elif x == 'N':
            n()
        elif x == 'o':
            o()
        elif x == 'O':
            o()
        elif x == 'p':
            p()
        elif x == 'P':
            p()
        elif x == 'q':
            q()
        elif x == 'Q':
            q()
        elif x == 'r':
            r()
        elif x == 'R':
            r()
        elif x == 's':
            s()
        elif x == 'S':
            s()
        elif x == 't':
            t()
        elif x == 'T':
            t()
        elif x == 'u':
            u()
        elif x == 'U':
            u()
        elif x == 'v':
            v()
        elif x == 'V':
            v()
        elif x == 'w':
            w()
        elif x == 'W':
            w()
        elif x == 'x':
            x()
        elif x == 'X':
            x()
        elif x == 'y':
            y()
        elif x == 'Y':
            y()
        elif x == 'z':
            z()
        elif x == 'Z':
            z()
        else:
            abre3()
        abre3()

def listener():

    # inicia el nodo
    rospy.init_node('articulaciones', anonymous=True)
```

```
    # esperamos a que se inicie todo
    time.sleep(3)

    # Se suscribe al canal chatter
    rospy.Subscriber("chatter", String, mover)

    # spin() evita que se detenga, crea un ciclo como un while(true)
en Java
    rospy.spin()


if __name__ == '__main__':
    listener()
```

13.4.2.2 controlSHVoz.py

El código es el mismo que el anterior, cambiando la siguiente parte correspondiente al método listener:

```
    # Se suscribe al canal del recognizer del Pocketsphinx
    rospy.Subscriber("/recognizer/output", String, mover)
```

13.4.3 Páginas web

13.4.3.1 Saluda.html

Esta versión no modifica ni el texto enviado ni el recibido.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />

<script type="text/javascript"
src="http://cdn.robotwebtools.org/EventEmitter2/current/eventemi
tter2.min.js"></script>
<script type="text/javascript"
src="http://cdn.robotwebtools.org/roslibjs/current/roslib.min.js
"></script>

<script type="text/javascript" type="text/javascript">
  // Conexión con ROS, aquí podemos poner la URL y el puerto al
que se debe dirigir.
  // Si es en local, con 'ws://localhost:9090' funcionará bien.
  // -----------------

  var ros = new ROSLIB.Ros({
    url : 'ws://localhost:9090'
  });

  var mensajeAnterior = ''; //Para evitar duplicados comprobamos
que el mensaje nuevo no es repetición del anterior.
```

```
  var estaEsperando = 'no'; //Eso servirá para que no se abran
nuevos hilos y optimizar la página.


  // Publicando un Topic para listenerNEsp.py
  // ------------------

  function habla(){
     var menTop = new ROSLIB.Topic({
     ros : ros,
     name : '/chatter', //Publicamos en chatter
     messageType : 'std_msgs/String' //Publicaremos un String
     });

     var texto = document.getElementById("caja").value;
//Cogemos el texto de caja

     var mensaje = new ROSLIB.Message({data : texto}); //Prepara
el mensaje a publicar.

     menTop.publish(mensaje); //Publica el mensaje.
     document.getElementById("caja").value = '';
     document.getElementById("caja").submit();
  }

  // Suscribirse a un Topic, código adaptado desde un ejemplo
pero aún no aprovechado, será lo próximo.
  // ----------------------

  var listener = new ROSLIB.Topic({
    ros : ros,
    //name : '/chatter', //Para que coincida con el de talker.py
para la prueba, en la versión final usaría otro.
     name: 'recognizer/output', //recognizer de Pocketsphinx
publica en este canal
    messageType : 'std_msgs/String'
  });

  listener.subscribe(function(message) {
    console.log('Received message on ' + listener.name + ': ' +
message.data);
    listener.unsubscribe();
  });

  // Empieza a escuchar lo que dice el robot. Método de Prueba.
  // ----------------------
  function escucharP(){
     document.getElementById('escuchado').innerHTML +=
"Probando</p><p>";
     document.getElementById('Boton').scrollIntoView();
  }

  // Activa la función de escuchar en márgenes de un segundo
para evitar trabas.
  // ----------------------
  function invocaEscuchar(){
```

```
      setInterval(function(){escuchar()},1000); //Para no crear
un bucle infinito que nos traba la aplicación, estamos iniciando
escuchas cada segundo.
      //escuchar();
  }

  // Si no hay un hilo abierto, abre un nuevo hilo para escuchar
en cuanto llegue el mensaje y escribirlo en la página.
  // ----------------------
  function escuchar(){

      if (estaEsperando == 'no'){

      estaEsperando = 'si';
      //La siguiente línea está para comprobar que se cambia el
nombre de la variable y que el método de optimización funciona
correctamente, es decir, que no se están iniciando nuevos hilos
de escucha. Descomentar para probarlo.
      //document.getElementById('escuchado').innerHTML +=
"Probando "+estaEsperando+"<br>";

      //Establecemos de dónde vamos a escuchar
      var listener = new ROSLIB.Topic({
    ros : ros,
    name : '/chatter', //Para hacer pruebas estamos escuchando
de la misma fuente en la que escribimos.
    messageType : 'std_msgs/String'
      });

      //Nos suscribimos al nodo que hemos indicado
      listener.subscribe(function(message) {
    console.log('Recibido mensaje de ' + listener.name + ': ' +
message.data);
      //Las escuchas que se abrieron anteriormente cogerán el
mismo mensaje cuando se reciba, de esta forma evitamos que se
dupliquen al escribirse.
      if (message.data != mensajeAnterior) {
            //document.getElementById('escuchado').innerHTML +=
'Mensaje recibido en ' + listener.name + ': ' +
message.data+'<br>'; //Escribimos el nuevo mensaje en la página
con los datos del canal escuchado.
            document.getElementById('escuchado').innerHTML +=
'Mensaje: ' + message.data+'<br>'; //Escribimos el nuevo mensaje
en la página.
            mensajeAnterior = message.data; //Actualizamos el
mensaje anterior.
      }
      listener.unsubscribe(); //Se acaba el hilo.
      estaEsperando = 'no'; //Como hemos cerrado el hilo ya
podemos abrir otro para el próximo mensaje.
      //La siguiente línea está para comprobar que se cambia el
nombre de la variable y que el método de optimización funciona
correctamente. Descomentar para probarlo.
      //document.getElementById('escuchado').innerHTML +=
"Probando "+estaEsperando+"<br>";
      document.getElementById('Boton').scrollIntoView();
//Bajamos el scroll automáticamente.
```

```
      });
      } //del if estaEsperando == no
  }

    function pulsaIntro(e){
     var key=e.keyCode || e.which;
     if (key==13){
           habla();
     }
  }

</script>
</head>

<body>
  <h1>Intérprete Robot</h1>

  <input type="button" name="BotonE" value="Empezar a escuchar
al robot" onclick="invocaEscuchar()">
                <p id="escuchado"></p>
  <p>Escribe lo que quieras y dale al botón para que el robot
hable.</p>
  <input type="text" id="caja" name="caja" value="" size=70
onkeypress="pulsaIntro(event)"><br>
  <input type="button" name="Boton" id="Boton" value="Haz que
hable el robot" onclick="habla()">
</body>
</html>
```

13.4.3.2 Greet.html

Igual que el anterior, pero con los textos que se muestran en pantalla en inglés:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />

<script type="text/javascript"
src="http://cdn.robotwebtools.org/EventEmitter2/current/eventemi
tter2.min.js"></script>
<script type="text/javascript"
src="http://cdn.robotwebtools.org/roslibjs/current/roslib.min.js
"></script>

<script type="text/javascript" type="text/javascript">
  // Conexión con ROS, aquí podemos poner la URL y el puerto al
que se debe dirigir.
  // Si es en local, con 'ws://localhost:9090' funcionará bien.
  // -----------------

  var ros = new ROSLIB.Ros({
    url : 'ws://localhost:9090'
  });
```

```
  var mensajeAnterior = ''; //Para evitar duplicados comprobamos
que el mensaje nuevo no es repetición del anterior.
  var estaEsperando = 'no'; //Eso servirá para que no se abran
nuevos hilos y optimizar la página.


  // Publicando un Topic para listenerNEsp.py
  // ------------------

  function habla(){
     var menTop = new ROSLIB.Topic({
     ros : ros,
     name : '/chatter', //Publicamos en chatter
     messageType : 'std_msgs/String' //Publicaremos un String
     });

     var texto = document.getElementById("caja").value;
//Cogemos el texto de caja

     var mensaje = new ROSLIB.Message({data : texto}); //Prepara
el mensaje a publicar.

     menTop.publish(mensaje); //Publica el mensaje.
     document.getElementById("caja").value = '';
     document.getElementById("caja").submit();
  }

  // Suscribirse a un Topic, código adaptado desde un ejemplo
pero aún no aprovechado, será lo próximo.
  // ----------------------

  var listener = new ROSLIB.Topic({
    ros : ros,
    //name : '/chatter', //Para que coincida con el de talker.py
para la prueba, en la versión final usaría otro.
     name: 'recognizer/output', //recognizer de Pocketsphinx
publica en este canal
    messageType : 'std_msgs/String'
  });

  listener.subscribe(function(message) {
    console.log('Received message on ' + listener.name + ': ' +
message.data);
    listener.unsubscribe();
  });

  // Empieza a escuchar lo que dice el robot. Método de Prueba.
  // ----------------------
  function escucharP(){
     document.getElementById('escuchado').innerHTML +=
"Probando</p><p>";
     document.getElementById('Boton').scrollIntoView();
  }

  // Activa la función de escuchar en márgenes de un segundo
para evitar trabas.
  // ----------------------
```

```
  function invocaEscuchar(){
      setInterval(function(){escuchar()},1000); //Para no crear
un bucle infinito que nos traba la aplicación, estamos iniciando
escuchas cada segundo.
      //escuchar();
  }

  // Si no hay un hilo abierto, abre un nuevo hilo para escuchar
en cuanto llegue el mensaje y escribirlo en la página.
  // ----------------------
  function escuchar(){

      if (estaEsperando == 'no'){

      estaEsperando = 'si';
      //La siguiente línea está para comprobar que se cambia el
nombre de la variable y que el método de optimización funciona
correctamente, es decir, que no se están iniciando nuevos hilos
de escucha. Descomentar para probarlo.
      //document.getElementById('escuchado').innerHTML +=
"Probando "+estaEsperando+"<br>";

      //Establecemos de dónde vamos a escuchar
      var listener = new ROSLIB.Topic({
    ros : ros,
    name : '/chatter', //Para hacer pruebas estamos escuchando
de la misma fuente en la que escribimos.
    messageType : 'std_msgs/String'
      });

      //Nos suscribimos al nodo que hemos indicado
      listener.subscribe(function(message) {
    console.log('Recibido mensaje de ' + listener.name + ': ' +
message.data);
      //Las escuchas que se abrieron anteriormente cogerán el
mismo mensaje cuando se reciba, de esta forma evitamos que se
dupliquen al escribirse.
      if (message.data != mensajeAnterior) {
            //document.getElementById('escuchado').innerHTML +=
'Mensaje recibido en ' + listener.name + ': ' +
message.data+'<br>'; //Escribimos el nuevo mensaje en la página
con los datos del canal escuchado.
            document.getElementById('escuchado').innerHTML +=
'Message: ' + message.data+'<br>'; //Escribimos el nuevo mensaje
en la página.
            mensajeAnterior = message.data; //Actualizamos el
mensaje anterior.
      }
      listener.unsubscribe(); //Se acaba el hilo.
      estaEsperando = 'no'; //Como hemos cerrado el hilo ya
podemos abrir otro para el próximo mensaje.
      //La siguiente línea está para comprobar que se cambia el
nombre de la variable y que el método de optimización funciona
correctamente. Descomentar para probarlo.
      //document.getElementById('escuchado').innerHTML +=
"Probando "+estaEsperando+"<br>";
```

```
      document.getElementById('Boton').scrollIntoView();
//Bajamos el scroll automáticamente.
      });
      } //del if estaEsperando == no
  }

    function pulsaIntro(e){
     var key=e.keyCode || e.which;
     if (key==13){
           habla();
     }
  }

</script>
</head>

<body>
  <h1>Robot Interpreter</h1>

  <input type="button" name="BotonE" value="Begin hearing from
the robot" onclick="invocaEscuchar()">
                <p id="escuchado"></p>
  <p>Write what you want and press the button to make the robot
talk.</p>
  <input type="text" id="caja" name="caja" value="" size=70
onkeypress="pulsaIntro(event)"><br>
  <input type="button" name="Boton" id="Boton" value="Make the
robot talk" onclick="habla()">
</body>
</html>
```

13.4.3.3 saludaTraduce.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />

<script type="text/javascript"
src="http://cdn.robotwebtools.org/EventEmitter2/current/eventemi
tter2.min.js"></script>
<script type="text/javascript"
src="http://cdn.robotwebtools.org/roslibjs/current/roslib.min.js
"></script>

<script type="text/javascript" type="text/javascript">
  // Conexión con ROS, aquí podemos poner la URL y el puerto al
que se debe dirigir.
  // Si es en local, con 'ws://localhost:9090' funcionará bien.
  // -----------------

  var ros = new ROSLIB.Ros({
    url : 'ws://localhost:9090'
  });
```

```
  var mensajeAnterior = ''; //Para evitar duplicados comprobamos
que el mensaje nuevo no es repetición del anterior.
  var estaEsperando = 'no'; //Eso servirá para que no se abran
nuevos hilos y optimizar la página.
  var traduccion = '';


  // Publicando un Topic para listenerNEsp.py
  // ------------------

  function habla(){
     var menTop = new ROSLIB.Topic({
     ros : ros,
     name : '/chatter', //Publicamos en chatter
     messageType : 'std_msgs/String' //Publicaremos un String
     });

     traducirEsEn();

     var mensaje = new ROSLIB.Message({data : traduccion});
//Prepara el mensaje a publicar.

     menTop.publish(mensaje); //Publica el mensaje.
     document.getElementById("caja").value = '';
     document.getElementById("caja").submit();
  }

  // Suscribirse a un Topic, código adaptado desde un ejemplo
pero aún no aprovechado, será lo próximo.
  // ----------------------

  var listener = new ROSLIB.Topic({
    ros : ros,
    //name : '/chatter', //Para que coincida con el de talker.py
para la prueba, en la versión final usaría otro.
     name: 'recognizer/output', //recognizer de Pocketsphinx
publica en este canal
    messageType : 'std_msgs/String'
  });

  listener.subscribe(function(message) {
    console.log('Received message on ' + listener.name + ': ' +
message.data);
    listener.unsubscribe();
  });

  // Empieza a escuchar lo que dice el robot. Método de Prueba.
  // ----------------------
  function escucharP(){
     document.getElementById('escuchado').innerHTML +=
"Probando</p><p>";
     document.getElementById('Boton').scrollIntoView();
  }

  // Activa la función de escuchar en márgenes de un segundo
para evitar trabas.
  // ----------------------
```

```
  function invocaEscuchar(){
      setInterval(function(){escuchar()},1000); //Para no crear
un bucle infinito que nos traba la aplicación, estamos iniciando
escuchas cada segundo.
      //escuchar();
  }

  // Si no hay un hilo abierto, abre un nuevo hilo para escuchar
en cuanto llegue el mensaje y escribirlo en la página.
  // ----------------------
  function escuchar(){

      if (estaEsperando == 'no'){

      estaEsperando = 'si';
      //La siguiente línea está para comprobar que se cambia el
nombre de la variable y que el método de optimización funciona
correctamente, es decir, que no se están iniciando nuevos hilos
de escucha. Descomentar para probarlo.
      //document.getElementById('escuchado').innerHTML +=
"Probando "+estaEsperando+"<br>";

      //Establecemos de dónde vamos a escuchar
      var listener = new ROSLIB.Topic({
    ros : ros,
    name : '/chatter', //Para hacer pruebas estamos escuchando
de la misma fuente en la que escribimos.
    messageType : 'std_msgs/String'
      });

      //Nos suscribimos al nodo que hemos indicado
      listener.subscribe(function(message) {
    console.log('Recibido mensaje de ' + listener.name + ': ' +
message.data);
      //Las escuchas que se abrieron anteriormente cogerán el
mismo mensaje cuando se reciba, de esta forma evitamos que se
dupliquen al escribirse.
      if (message.data != mensajeAnterior) {
            //document.getElementById('escuchado').innerHTML +=
'Mensaje recibido en ' + listener.name + ': ' +
message.data+'<br>'; //Escribimos el nuevo mensaje en la página
con los datos del canal escuchado.
            document.getElementById('escuchado').innerHTML +=
'Mensaje original: ' + message.data+'<br>'; //Escribimos el
nuevo mensaje en la página.
            mensajeAnterior = message.data; //Actualizamos el
mensaje anterior.
            traducirEnEs(); //traducimos
            document.getElementById('escuchado').innerHTML +=
'Mensaje traducido: ' + traduccion+'<br><br>'; //Escribimos el
nuevo mensaje en la página.
      }
      listener.unsubscribe(); //Se acaba el hilo.
      estaEsperando = 'no'; //Como hemos cerrado el hilo ya
podemos abrir otro para el próximo mensaje.
```

```
      //La siguiente línea está para comprobar que se cambia el
nombre de la variable y que el método de optimización funciona
correctamente. Descomentar para probarlo.
      //document.getElementById('escuchado').innerHTML +=
"Probando "+estaEsperando+"<br>";
      document.getElementById('Boton').scrollIntoView();
//Bajamos el scroll automáticamente.
      });
      } //del if estaEsperando == no
  }

    function pulsaIntro(e){
      var key=e.keyCode || e.which;
      if (key==13){
            habla();
      }
  }

    function traducirEsEn(){

      var texto = document.getElementById("caja").value;
//Cogemos el texto de caja

            var
xmlDoc=loadXMLDoc("http://syslang.com/?src=es&dest=en&text="+tex
to+"&email=dark_enzan@yahoo.es&password=bass14"); //Enviamos el
texto al traductor
            traduccion =
xmlDoc.getElementsByTagName("translation")[0].childNodes[0].node
Value + "<br>"; //Guardamos la traducción
  }

      function traducirEnEs(){

      var texto = mensajeAnterior; //Cogemos el texto del mensaje
anterior

            var
xmlDoc=loadXMLDoc("http://syslang.com/?src=en&dest=es&text="+tex
to+"&email=dark_enzan@yahoo.es&password=bass14"); //Enviamos el
texto al traductor
            traduccion =
xmlDoc.getElementsByTagName("translation")[0].childNodes[0].node
Value + "<br>"; //Guardamos la traducción
  }

</script>
</head>

<body>
  <h1>Intérprete Robot</h1>

  <input type="button" name="BotonE" value="Empezar a escuchar
al robot" onclick="invocaEscuchar()">
                  <p id="escuchado"></p>
  <p>Escribe lo que quieras y dale al botón para que el robot
hable.</p>
```

```
  <input type="text" id="caja" name="caja" value="" size=70
onkeypress="pulsaIntro(event)"><br>
  <input type="button" name="Boton" id="Boton" value="Haz que
hable el robot" onclick="habla()">
</body>
</html>
```