Universidad Carlos III de Madrid
www.uc3m.es

Grado en Ingeniería Electrónica Industrial y Automática

2015 - 2016

Trabajo Fin de Grado

"Sistema de apoyo a la terapia vocal para un robot social"

Diego Álvarez Fernández

Tutor:

José Carlos Castillo Montoya

Leganés, septiembre 2016

II

TRABAJO FIN DE GRADO

Sistema de apoyo a la terapia vocal para un robot social

Autor: Diego Álvarez Fernández

Tutor: José Carlos Castillo Montoya

EL TRIBUNAL

Presidente:

Vocal:

Secretario:

Realizado el acto de defensa y lectura del Trabajo Fin de Grado el día …… de …….. de …… en …… en la Escuela Politécnica Superior de la Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de:

VOCAL

SECRETARIO **PRESIDENTE**

Agradecimientos

Parece que fue ayer cuando entré por las puertas de esta universidad, pensando en lo largos que se me iban a hacer cuatro años más estudiando. Y en un abrir y cerrar de ojos, estoy terminando este Trabajo Fin de Grado y a nada de tener mi grado. En estos cuatro años, he vivido un montón de experiencias que han definido la persona que soy ahora.

Me gustaría dar las gracias en primer lugar a mi tutor José Carlos. Gracias por aguantar mis molestas hordas de preguntas y e-mails, y por guiarme y aconsejarme con cada duda que tenía. Ha sido un verdadero placer tenerte como tutor durante todo este tiempo. Además, quiero agradecer al grupo de RoboticsLab por estar ahí siempre que los necesitaba. Sin ellos, seguro que no hubiese llegado ni la mitad de lejos con mi Trabajo.

Quiero agradecer a mi familia todo el apoyo y ayuda que me han dado. No hay nadie mejor que ellos que sepan lo que he peleado por esto, y siempre han estado ahí para animarme y echarme una mano. Nada de esto hubiese sido posible sin ellos.

A mis compañeros de la Universidad: Juanfran, Vecino, Cristian, Román, Paco, Fer, Enrique, Alex, Carmen, Paula y Teresa. No quiero olvidarme de ninguno, porque todos han puesto su granito de arena durante estos cuatro años. Sin duda, son una de las mejores cosas que me llevo de toda esta experiencia.

A Juan, por todas esas tardes pasadas en el laboratorio usándote como conejillo de indias con mi proyecto. Has estado ahí para todo lo que he necesitado, y te has comportado como un verdadero amigo.

A todos los que se molestaron en hacerse el viaje a Leganés para echarme una mano: Alex, Jorge, Almudena, Jacobo, Teresa y mi hermanita Ana.

Y, por último, quiero dar las gracias a Teresa. Gracias por estos casi 4 años ayudándome y apoyándome, incluso cuando ni lo necesitaba. Has sido mi apoyo durante toda mi etapa en la Universidad, y seguramente, todo ha sido mucho más fácil gracias a ti.

Muchas gracias a todos.

Resumen

Hoy en día, no es común encontrar terapias robotizadas enfocadas en la apraxia del habla para pacientes con discapacidades motoras relacionadas con la demencia, Alzheimer, o que hayan sufrido un ictus. El objetivo de este proyecto es el desarrollo de un software que pueda integrarse en un robot social para la realización de dichas terapias. Con esto, se pretende conseguir crear una terapia novedosa para el paciente y que requiera un menor grado de atención para el terapeuta, ya que el paciente estará "jugando" con el robot.

En este documento se muestra el Trabajo Fin de Grado titulado "Sistema de apoyo a la terapia vocal para un robot social", realizado en el grupo de investigación RoboticsLab de la Universidad Carlos III de Madrid, en la Escuela Politécnica Superior de Leganés.

En este proyecto haremos uso de un clasificador para determinar qué pose está vocalizando el usuario. Tomamos los datos (imágenes y puntos en 3D) desde un sensor Kinect, que está integrado dentro de la arquitectura del robot Mini. El manejo de los datos provenientes del sensor es realizado por un que nos da 18 puntos con los que se caracteriza la boca del usuario. Dichos puntos se utilizan en la fase de entrenamiento del sistema o para realizar el ejercicio de la terapia.

La terapia se realizará utilizando el robot Mini. El robot será capaz de proponer al usuario una pose y corregirle en el caso de que realice mal la pronunciación. Además, aprovechando la capacidad de síntesis de voz del robot Mini, se guiará al usuario durante los ejercicios, o se le felicitará si vocaliza correctamente.

Palabras clave: Robótica, Terapias de Rehabilitación, Aprendizaje Automático, Apraxia del habla, ROS, RGB-D.

Abstract

Nowadays, there are not many robotic therapies focused on speech apraxia in patients with motor disabilities related dementia, Alzheimer or who have suffered a stroke. The aim of this project is the realization of a software that can be implemented in a social robot in order to perform such therapies. This is intended to create a more pleasant therapy for the patient, which requires a lesser degree of attention to the therapist, as the patient will be " playing" with the robot.

This document describes the Bachelor Thesis entitled “Sistema de apoyo a la terapia vocal para un robot social”, made in the research group called RoboticsLab of the Carlos III University of Madrid, in the Polytechnic School in Leganes.

In this project we will use a classifier to determine which vowel is the user pronouncing. We take data (images and 3D points) with a Kinect sensor, which is integrated into the architecture of the Mini robot. Management data from the sensor is performed by a node that provides 18 reference points of how the user's mouth is characterized. These points can be used to train the classifier or to exercise therapy.

The therapy is carried out with the support of the Mini robot. The robot will be able to offer the user a vowel and correct him. In addition, the robot may indicate the patient to open the mouth more or less, or congratulate if the vocalization is correct.

Keywords: Robotics, Rehabilitation Therapies, Machine Learning, Speech apraxia, ROS, RGB-D.

Índice General

Índice de figuras

Índice de tablas

1. Introducción

Hoy en día, no es común encontrar terapias robotizadas enfocadas en la apraxia del habla para pacientes con discapacidades motoras relacionadas con la demencia, Alzheimer, o que hayan sufrido un ictus. El objetivo de este proyecto es el desarrollo de un software que pueda integrarse en un robot social para la realización de dichas terapias. Con esto, se pretende conseguir crear una terapia novedosa para el paciente y que requiera un menor grado de atención para el terapeuta, ya que se diseñará como un juego con el robot.

Este proyecto ha sido realizado dentro del grupo de investigación RoboticsLab de la Universidad Carlos III de Madrid. Para su realización, se utilizó el robot MINI. Se ha tomado como punto de inicio el proyecto "Diseño e implementación de un sistema de reconocimiento vocal a través de información RGB-D" de Alfonso Conti Morera, por el cual se puede entrenar un estimador para la clasificación de imágenes off-line. En este trabajo se ha cambiado el software utilizado en el anterior, que corresponde a la parte posterior a la detección de la boca del usuario, con lo que se pretende mejorar los resultados obtenidos. También se ha modificado el clasificador para su uso vía on-line, y se ha diseñado una interfaz para la interacción robot-usuario durante la terapia.

Los siguientes puntos de la introducción serán: una descripción del contexto y la motivación que han llevado a la realización de este proyecto y los objetivos a realizar.

1.1 Contexto

Podemos definir la apraxia del habla como un trastorno neurológico que se caracteriza por la afección en la planificación o programación de los movimientos oro-faciales necesarios para realizar los sonidos del habla [1]. El cerebro carece de la coordinación necesaria para articular las palabras, aunque el sujeto tiene voluntad de hacerlo. Este trastorno se produce por una lesión en el hemisferio izquierdo. Puede haber distintos desencadenantes: tumor cerebral, ictus, Alzheimer…

Dentro de la apraxia del habla podemos diferenciar la de tipo temporal, relacionada con la desconexión entre el sistema fonológico y fonético y la de tipo espacial, que se explica en la alteración en las habilidades motoras finas. En nuestro caso, nos centraremos en la apraxia del habla de tipo temporal. Podemos ver las diferencias más notables entre ambos tipos en la Tabla 1.1. Como resultado, el paciente

tendrá errores articulatorios inconsistentes, ensayos articulatorios (movimiento de la boca previo al habla producido por errores de vocalización), disprosodia (emisión de fonemas de manera descontrolada) y una disminución en la velocidad del habla [1].

Tabla 1.1: Tipos de apraxia del habla

Apraxia del habla de tipo temporal

- Esfuerzo, ensayo y error, movimientos articulatorios tentativos e intentos de autocorrección.
- Disprosodia.
- Inconsistencia articulatoria sobre repetidas producciones del mismo enunciado
- Dificultad evidente para iniciar los enunciados
- Disociación automática – voluntaria

Apraxia del habla de tipo espacial

- Desintegración fonética (distorsión de sonidos severa y variable)
- Inconsistencia articulatoria sobre repetidas producciones del mismo enunciado
- No presentan dificultad para iniciar.
- Falta de conciencia del defecto.
- En la mayoría de los casos coexiste con afasia severa

Podemos definir varios tipos de terapia que se emplean actualmente en la apraxia del habla:

-*Programas de intervención basados en el control motor*: Estos programas consisten en producir fonemas y secuencias de fonemas mediante movimientos voluntarios, conscientes, controlados y precisos. Su objetivo es automatizar los movimientos para poder así realizarlos de forma inconsciente. Se busca con ellos la mejora del control involuntario del habla [2].

-*Programas de intervención basados en sistemas aumentativos*: Mediante este método se pretende utilizar varios canales de entrada para la mejora de los resultados de la terapia. Se utiliza información auditiva junto a información visual, la cual refuerza a la primera. El uso de imágenes ayuda a recordar la pronunciación de palabras complicadas o largas [2].

-*Programas basados en la melodía*: estas terapias se realizan en pacientes que conservan la comprensión auditiva del lenguaje. El paciente debe imitar las melodías que le propone el logopeda. En estas melodías se remarcan las sílabas tónicas de las palabras, dándole ritmo a la melodía [2].

En este proyecto se diseñará una terapia basada en el control motor, pero apoyada con imágenes para aumentar su eficacia. Se usará el robot Mini para corregir al paciente en la vocalización de diferentes sonidos del habla. Además, utilizando imágenes en la pantalla que posee, se dará un refuerzo a la terapia, indicando cómo debe vocalizar el usuario para conseguir hacerlo con éxito.

1.2 Motivación

El uso de robots en terapias de rehabilitación está siendo uno de los temas principales de investigación en la robótica actual. Su uso se ha incrementado notablemente en traumatología, donde el robot ayuda al usuario a mantener su propio peso durante la rehabilitación, o a mover de una determinada forma una extremidad de su cuerpo. Para este fin se utilizan exoesqueletos como el robot H2, que podemos ver en la Figura 1.2, el cual sirve para la rehabilitación de las extremidades inferiores, a la vez que monitoriza toda la información cerebral, medular y muscular. El uso de robots ayuda a los pacientes con parálisis parcial o total en sus piernas a poder llegar a realizar movimientos naturales con ellas [3].

Figura 1.1: Exoesqueleto H2

Visto el éxito y la acogida que pueden tener este tipo de terapias robotizadas, y lo esperanzador de sus resultados, se ha considerado la conveniencia de introducirlas en el tratamiento de la apraxia del habla. En este caso, hemos de trabajar con la repetición de ejercicios, ya que se debe acostumbrar al paciente a practicar movimientos vocales hasta que sean naturales para él y consiga realizarlos de forma automática. Para este tipo de pacientes, las actuales terapias pueden llegar a ser aburridas o poco amenas, al limitarse a la repetición de ejercicios propuestos por un

logopeda [4], y esto condiciona el éxito de sus resultados. No existen hoy en día robots que ayuden al terapeuta en estos ejercicios, lo que hace de este proyecto algo innovador.

Con la utilización de un robot social se mejoraría la capacidad de atención de los pacientes, puesto que ofrece una forma llamativa y novedosa de realizar los ejercicios. Al captar una mayor atención por parte del usuario, ganaría en eficacia y consecución de resultados, ya que los ejercicios se realizarían con mayor ímpetu y ganas, y el nivel de cansancio y frustración descendería notablemente. Además, se conseguiría ampliar la cantidad y variedad de ejercicios y tratamientos. Un robot añadiría ciertos recursos que un logopeda por sí solo no tendría, como una pantalla dinámica que estimulase al paciente. Así se conseguiría un refuerzo visual en los ejercicios, y se podrían dar instrucciones tanto por pantalla como por gestos.

El diseño de estas terapias nos permite estudiar a fondo el uso del Aprendizaje Automático para la creación de nuestro software. Con este recurso, podemos manejar gran cantidad de datos en poco tiempo. Nos permite profundizar en el uso de clasificadores para la gestión de datos (puntos) y el ajuste de estos para mejorar los resultados obtenidos. El Aprendizaje Automático es una técnica muy valorada por matemáticos, expertos en estadística e ingenieros, lo que da al proyecto un mayor grado de innovación, ya que su uso hoy en día está en auge.

En conclusión, la motivación de este proyecto es conseguir desarrollar nuevas terapias para el tratamiento de la apraxia del habla mediante la creación de un software capaz de ser integrado en un robot social.

Esto conllevaría el diseño de un software que tomase datos del exterior, otro que se encargase del procesamiento de datos mediante Aprendizaje Automático, y un último que definiera la interfaz para el desarrollo de la actividad. La interfaz sería la parte más importante de nuestra terapia, donde se darían las instrucciones al paciente y se le estimularía con imágenes o gestos del robot para mejorar los resultados. Se debe crear un ejercicio coherente, en el cual no se pierda la atención del paciente en ningún momento, y le haga realizar los ejercicios de la manera más eficiente posible.

1.3 Objetivos del trabajo

En esta sección vamos a definir los objetivos de este trabajo:

1. Estudio del trabajo previo. Este proyecto parte de un proyecto llamado “Diseño e implementación de un sistema de reconocimiento vocal a través de información RGB-D”, por el cual debemos estudiar qué partes de él son aptas

para nuestro proyecto, cuales tras previa modificación se pueden utilizar, y que partes no son válidas para nuestro trabajo.

2. Estudio de las técnicas de Aprendizaje Automático y su aplicación en la terapia desarrollada. Se buscará la implementación de un clasificador para que el robot social pueda distinguir que pose vocal está realizando el usuario.

3. Diseño e implementación de un programa para entrenar el clasificador seleccionado. Debemos crear un software capaz de tomar los datos de un módulo de detección previo. Estos datos serán modificados de tal manera que sean aptos para entrenar nuestro clasificador. Los datos de entrenamiento serán guardados en ficheros para su uso posterior si se desea ampliar el número de muestras de entrenamiento.

4. Diseño de ejercicios válidos para probar la eficacia de nuestro programa. Se debe diseñar un algoritmo que sea capaz de hacer de terapia para poder probar nuestro clasificador con usuarios reales. Esto será la base para poder construir en un futuro una terapia real que pueda utilizarse con pacientes con apraxia del habla.

1.4 Estructura del documento

En esta sección se describirán los contenidos que contienen los distintos capítulos que forman este documento. Esto hará más fácil al lector la comprensión y lectura del trabajo.

En el Capítulo 1, en el cual nos encontramos ahora, se redactará la Introducción. Se explicará al lector la motivación que ha hecho posible la realización de dicho proyecto, además de explicar el contexto que lo enmarca. Por último, se enumerarán y explicarán los objetivos que se quieren lograr con la realización del presente trabajo.

El Capítulo 2 estará enfocado en el Estado del arte. Se explicarán las terapias que existen hoy en día para la rehabilitación de pacientes que hayan sufrido enfermedades neurodegenerativas o accidentes cerebrovasculares. Además, enmarcaremos el estado en el que se encuentran las técnicas de aprendizaje automático empleadas en el presente proyecto. Para finalizar, se explicará en qué consiste el proyecto RobAlz y qué tecnologías han sido empleadas en el presente proyecto.

El Capítulo 3 tratará del Análisis y diseño del sistema. En dicho capítulo se describirán las restricciones del sistema, el entorno en el que se va a operar y los requisitos, funcionales y no funcionales, que debe cumplir el sistema desarrollado. Se

hablará también de los casos de uso y la metodología seguida para la realización del proyecto.

El Capítulo 4 hablará de la Implementación del sistema. Se describirá la integración de los componentes necesarios para desarrollar el proyecto, además de las herramientas que facilitan la realización de determinadas tareas durante la creación del sistema. Por último, se explicarán los procesos del entrenamiento del clasificador y de realización de la terapia.

En el Capítulo 5 se tratará la Evaluación del sistema. Se estudiarán los resultados obtenidos en las pruebas realizadas, detallando el comportamiento del sistema.

En el Capítulo 6 se hablará de la Planificación y el presupuesto previsto en dicho proyecto. En dicho capítulo se tratará la metodología de la planificación, y la planificación en sí misma. Se describirán los costes previstos y el presupuesto total estimado.

En el Capítulo 7 y último capítulo, nos enfocaremos en las Conclusiones y trabajos futuros. Se explicarán las conclusiones a las que se han llegado con la realización de este proyecto y las líneas futuras de investigación.

Al final del documento se añadirán distintos anexos (Apéndices A y B) que nos darán información complementaria del presente proyecto.

2. Estado del arte

Actualmente existen dos grupos de terapias para mejorar la calidad de vida de los afectados por una enfermedad neurodegenerativa o un accidente cerebrovascular: Manual, donde un terapeuta se encarga del mayor peso de la terapia; y automática o semiautomática, donde se incluyen las nuevas tecnologías para facilitar la tarea al terapeuta y mejorar la recuperación de los pacientes.

En este apartado haremos un estudio de las terapias convencionales, que son las únicas que tratan hasta el momento la apraxia del habla, y nos detendremos también en las terapias automáticas o robotizadas que se realizan en distintos ámbitos de la medicina, sobre todo los más cercanos a la rehabilitación de pacientes con demencia, Alzheimer o ictus, para posteriormente explicar las técnicas y tecnologías que pretendemos desarrollar.

2.1 Terapias convencionales

La apraxia del habla es un trastorno de programación motora del habla que disminuye la capacidad de planificar y ejecutar de forma voluntaria los movimientos adecuados para la articulación de sonidos coherentes. El proceso de recuperación de esta capacidad es una tarea larga y repetitiva. El problema viene principalmente en la organización de los movimientos en tiempo y espacio, y puede darse el caso de que el paciente pueda producir los sonidos de forma aislada, pero no sea capaz de organizar los movimientos en una secuencia de sonidos larga.

En la enfermedad del Alzheimer, en la demencia, o tras sufrir un ictus, es muy común perder la capacidad del habla. Las terapias asociadas a la logopedia se encargan de retrasar esta pérdida en el caso de enfermedades de deterioro cognitivo (Alzheimer, demencia…) y rehabilitar a los pacientes para recuperar la capacidad del habla en el caso de accidentes cerebrovasculares. Se caracterizan por realizar actividades de fluidez verbal: pedir al paciente que diga palabras que comiencen por un sonido, reconocer palabras o determinar sinónimos o antónimos. Los plazos generales del tratamiento rondan los 3 años [5]. Las terapias de logopedia dan muy buenos resultados en la mayoría de los pacientes, haciendo que estos logren alcanzar un nivel aceptable en sus capacidades del habla en unos plazos establecidos por el logopeda en el caso de accidentes cerebrovasculares. En los hospitales ya suele haber un departamento de logopedia que trabaja junto a médicos y psicólogos para mejorar la calidad de vida de los pacientes con lesiones cerebrovasculares o Alzheimer [6].

Otra interesante línea de investigación en este tipo de terapias es la musicoterapia, que se emplea como medio para conseguir determinados objetivos terapéuticos, ya sea para facilitar la comunicación, expresión o aprendizaje de los pacientes. Se utiliza con gente de edad avanzada, con afecciones neurológicas, enfermedades terminales, psiquiátricas o con problemas de dolores intensos. Este tipo de terapias resulta de utilidad en casos de pérdida de capacidad del habla, ya que se puede impulsar a los pacientes a emitir diferentes sonidos a partir de una melodía [7]. Neurólogos como el Dr. Oliver Sacks determinaron que la música tiene una gran capacidad de reorganizar la función cerebral cuando esta se encuentra alterada [8]. Se utiliza la música como elemento para hacer al paciente recordar determinados sucesos del pasado, y para activar zonas específicas del cerebro que ayudan en los procesos de rehabilitación.

De lo dicho hasta ahora se desprende la importancia de la labor de los logopedas. En este momento, su trabajo constituye la piedra angular de los tratamientos para la recuperación de la capacidad de hablar de manera consciente y voluntaria en aquellos que la tienen disminuida por motivos neurológicos. Por ello, intentaremos conseguir un ejercicio de terapia similar a los suyos, pero con el plus visual y motivador que puede aportar un robot social.

2.2 Terapias robotizadas

El creciente uso de la tecnología en hospitales y centros especializados en forma de terapias robotizadas para el tratamiento de distintas afecciones, y su excelente aceptación tanto por parte de los profesionales de la medicina y terapeutas como de los pacientes, hacen llamativo este campo para explorar su desarrollo.

En este contexto, trabajamos para la creación de un software que pueda implementarse en un robot para terapias contra la apraxia del habla en el Alzheimer, demencia, enfermedades neurodegenerativas o accidentes cerebrovasculares.

Dentro de las terapias robotizadas podemos distinguir aquellas que se centran en la rehabilitación de ancianos con demencia o Alzheimer, en las que se utilizan robots de aspecto amigable, con actividades que se centran en la estimulación cognitiva del paciente. Los ejercicios que se realizan con estos robots son supervisados por terapeutas y enfermeros en todo momento, y son ellos los que proponen al paciente las actividades a realizar.

Por un lado, podemos señalar al robot Paro (Nuka en español), que presenta una forma similar a una foca bebe, como podemos ver en la Figura 2.1 [9]. Este robot se usa en pacientes con demencia y su objetivo es reducir el estrés del paciente, mejorar sus

capacidades mentales mediante ejercicios simples y reforzar la motivación de sus usuarios. Con el uso de robots con apariencia animal, como Paro, se consiguen progresos y avances similares a las terapias con animales reales. En este caso, Paro es concebido como una mascota, lo que provoca una estimulación cognitiva en los pacientes que es muy beneficiosa para su terapia. El paciente debe prestar atención a las necesidades que le presenta el robot. Toda actividad realizada con el robot Paro debe de ser reforzada con la actuación de cuidadores y enfermeros [10].

Figura 2.1: Robot Paro

Este robot posee varios sensores para la interacción con el usuario, como un sensor de posición, sensores de tacto en la cabeza, debajo de la mandíbula, en los laterales y en la zona posterior, en la aleta delantera y en la aleta trasera. Posee dos sensores de luminosidad en la cabeza, un altavoz y dos micrófonos. Todo esto hace capaz al robot Paro de expresar emociones y sentimientos. Puede quejarse frente a golpes o malos tratos por parte del paciente, y puede mostrar agrado y felicidad si el paciente actúa de forma correcta.

Otro ejemplo del uso de robots en terapias cognitivas es el caso de Babyloid [11]. Como se puede ver en la Figura 2.2, este robot simula el aspecto de un bebe. Posee en sus brazos sensores de tacto, en la cabeza sensores de luz y piroeléctricos, y en el cuerpo más sensores de tacto, un micrófono y una cámara.

Figura 2.2: Apariencia del robot Babyloid

El uso de robots con apariencia de peluche en terapias cognitivas se denomina "doll therapy". Se ha demostrado que el empleo de estas terapias disminuye la velocidad a la que progresa la demencia en los pacientes expuestos a dichos robot. El robot debe tener un aspecto amigable e inofensivo, para que, de esta manera, sea aceptado por el usuario.

La terapia en este caso consistirá en cuidar al robot como si de un bebe se tratase, respondiendo a las reacciones que éste presente, lo que requiere captar y mantener la atención del paciente. Babyloid puede expresar tanto dolor como sueño o alegría [11]. En la Figura 2.3 se muestra un resumen de las reacciones que puede tener el robot Babyloid.

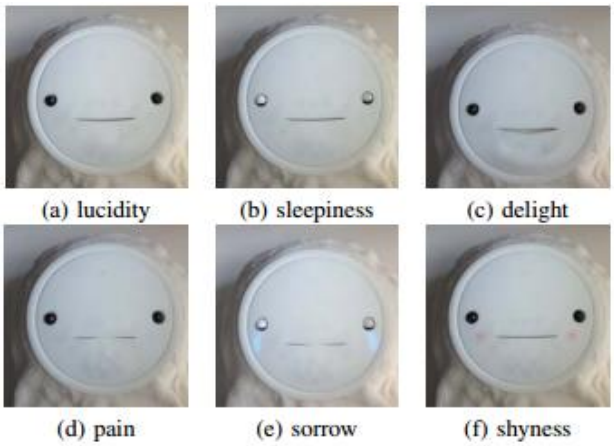


Figura 2.3: Reacciones del robot Babyloid

Estos dos robots pueden enmarcarse dentro de los usados en "doll therapy". Despiertan en el paciente la necesidad de cuidarlos y para ello, realizar las actividades que les propone el terapeuta. Existe otro tipo de robots más interactivos. Un ejemplo es el robot ELDERTOY, que podemos observar en la Figura 2.4 [12]. Este robot puede considerarse un juguete para mayores de 60 años, con el cual pueden realizarse terapias

cognitivas para el tratamiento de demencias avanzadas. Con este tipo de robots se consigue estimular las capacidades del usuario y mitigar las perdidas cognitivas propias de la demencia.

ELDERTOY posee una pantalla táctil donde el usuario puede jugar y realizar las terapias cognitivas, y actuadores dan expresividad al robot frente a las actuaciones del usuario. Mediante la manipulación directa de ELDERTOY, ya sea hablando con él o actuando con su pantalla, es posible desarrollar terapias que trabajen la coordinación, memoria, calculo o el habla. Por otra parte, se puede usar como una videoconsola, proyectando la imagen por un monitor o un proyector, por lo que se puede utilizar cualquier imagen, video o juego interactivo que desarrolle las capacidades físicas y cognitivas del paciente.

Figura 2.4: Robot ELDERTOY

ELDERTOY permite la interacción por numerosos canales, como por su pantalla táctil, por voz o por gestos. La pantalla es el soporte para los juegos y para añadir expresividad al diseño final del juguete. Posee un giroscopio que nos da la orientación y puede usarse en determinados juegos. Su cámara, micrófono y altavoz permiten que el usuario pueda identificarse con el robot, dependiendo de su expresión. ELDERTOY es capaz de reconocer voces y detectar la presencia de personas a su alrededor [12].

2.3 Aprendizaje Automático

El aprendizaje automático es una rama de la ingeniería de computación que consiste en el estudio de reconocimiento de patrones y teorías de aprendizaje computacional en la inteligencia artificial. Se busca el estudio y construcción de algoritmos que puedan aprender y hacer predicciones a partir de unos datos dados. Con ello, se consigue que las máquinas aprendan por sí mismas, el desarrollo de un motor

de habilidades cognitivas a partir de observaciones o de experimentaciones y la capacidad de organizar dicho conocimiento. A partir de esto, una máquina es capaz de realizar deducciones lógicas partiendo de modelos estadísticos, como escoger muestras similares, realizar clasificaciones u otros procesos [13].

El aprendizaje automático (o también llamado Machine Learning) se basa principalmente en dos técnicas: aprendizaje supervisado y aprendizaje no supervisado. En el caso del aprendizaje supervisado, se crea una relación entre las entradas y salidas del sistema, es decir, se crea una especie de etiquetas o clasificaciones para las entradas, lo que establecerá la salida. En el caso del aprendizaje no supervisado, solamente se tienen en cuenta las entradas del sistema, sin realizar ningún tipo de etiquetado. Se basa en la realización de agrupaciones de las entradas para poder distinguir grupos de muestras similares [14].

Los usos de los algoritmos del aprendizaje automático son muy dispares. En la Tabla 2.1 podemos observar diferentes usos:

Tabla 2.1: Usos del aprendizaje automático

USOS DEL APRENDIZAJE AUTOMÁTICO		
Sitios web adaptativos	Recuperación de información	Locomoción de robots
Computación afectiva	Detección de fraude	Motores de búsqueda
Bio-informática	Marketing	Análisis de sentimiento
Interfaces cerebro-máquina	Percepción	Minería de secuencias
Informática química	Diagnósticos médicos	Ingeniería de software
Clasificación secuencias ADN	Procesamiento de lenguajes naturales	Reconocimiento de escritura y habla
Anatomía computacional	Traducción de lenguajes naturales	Análisis de stock
Reconocimiento de objetos	Optimización y metaheurística	Monitorización de la salud estructural
Seguridad tarjetas de crédito	Anuncios por internet	Reconocimiento de patrones sintácticos
Juegos de estrategia	Recomendación de sistemas	Economía

2.4 Proyecto RobAlz

El proyecto RobAlz, en el que este trabajo se incluye, nace de la colaboración de la organización FAE (Fundación de Alzheimer de España) y la Universidad Carlos III de Madrid. Tiene como objetivo explorar cuales son las posibilidades de la robótica en el ámbito de la demencia, concretamente, dentro de la enfermedad del Alzheimer. Con

ello se consigue, no solo ayudar a los enfermos de dicha enfermedad, sino también a los cuidadores, a los cuales se pretende liberar de cierta carga en los tratamientos de los pacientes [15].

Este proyecto se puede dividir en tres fases:

1. Definir los escenarios en los que el robot deba actuar.
2. Diseñar y construir el robot
3. Realizar experimentos y evaluaciones del robot con enfermos de Alzheimer y sus cuidadores en su entorno doméstico.

Actualmente este proyecto se encuentra en la segunda fase, elaborando dispositivos de robótica asistencial. Tras terminar esta fase, se deberá implantar el dispositivo en entornos reales para comprobar su funcionamiento. Se deberán valorar los modos de interacción humano-robot, su fiabilidad, su robustez y su diseño.

2.5 Tecnologías utilizadas

En la siguiente sección se describirán las diferentes herramientas y sensores utilizados para el desarrollo del presente proyecto.

2.5.1 Cámara Kinect

La cámara Kinect es un controlador desarrollado por Microsoft para videoconsola Xbox 360[1] que permite controlar la consola mediante gestos, comandos de voz, objetos e imágenes. Este sensor, que podemos ver en la Figura 2.5, consta de una barra horizontal de 23 centímetros conectada a una base circular con un eje de articulación de rótula. El dispositivo posee una cámara RGB, un sensor de profundidad, un micrófono de múltiples matrices y un procesador. Con todo esto, podemos capturar todo el cuerpo en 3D y podemos realizar reconocimientos faciales y vocales. El sensor de profundidad se compone de un proyector de infrarrojos y un sensor CMOS monocromo, lo cual permite ver la habitación en 3D con cualquier tipo de luz ambiental.

La geometría relativa entre el proyector de infrarrojos y la cámara infrarroja, así como el patrón de puntos infrarrojos proyectados, son conocidos. Si comparamos un punto observado en una imagen con un punto en el patrón proyectado, utilizando triangulación podemos reconstruir dicho punto en 3D. Debido a que el patrón de puntos

[1] Presentación de Xbox en E3: http://latimesblogs.latimes.com/technology/2009/06/microsofte3.html (ultimo acceso: 10/09/2016)

es relativamente aleatorio, la relación entre la imagen infrarroja y el patrón proyectado puede realizarse de una forma sencilla utilizando comparaciones de pequeñas zonas utilizadas, normalizadas por correlaciones cruzadas [16].

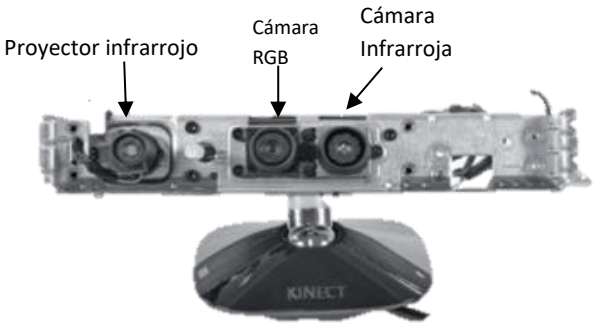


Figura 2.5: Proyector infrarrojo, cámara IR y cámara RGB de un sensor Kinect

Este dispositivo también se utiliza en otros ámbitos fuera de los videojuegos, debido a su bajo coste, su fácil configuración y sus amplias prestaciones. Se puede utilizar este sensor como herramienta para reconocimiento de objetos, reconocimiento facial y esqueleto humano, mapeado de entornos cerrados o para interacciones entre persona-robot. Este elemento es fundamental para el desarrollo del presente trabajo, ya que se utiliza para la captura de imágenes necesaria para el posterior proceso de detección facial.

2.5.2 Robot Mini

Mini es un robot social diseñado para personas de la tercera edad con deterioro cognitivo. Tiene un aspecto similar al robot Maggie [17], perteneciente al grupo de robots sociales del RoboticsLab de la universidad Carlos III de Madrid. De ahí también procede el robot Mini, el cual podemos ver en la Figura 2.6.

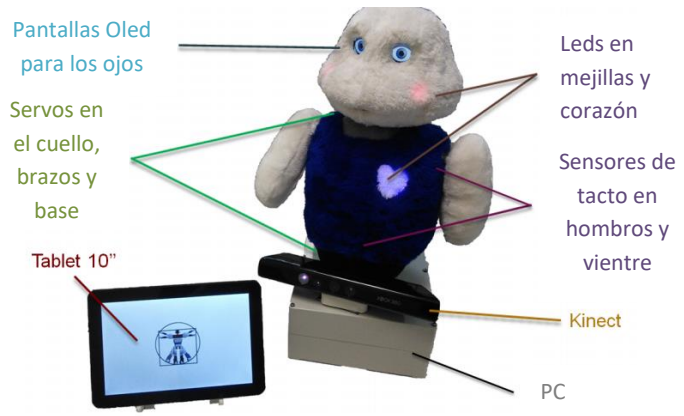


Figura 2.6: Robot Mini

Mini es un robot de sobremesa. Dispone de una batería interna para realizar un apagado controlado y así poder moverlo a otra ubicación cercana y evitar tener que relanzar el robot de nuevo. Posee un microprocesador i5 para el control central. Cuenta con 5 grados de libertad en brazos (dos), cabeza (dos) y en su base (uno) controlados por servos dynamixel. Para hacer a Mini más amigable de cara a la interacción con los usuarios, se le ha dado un recubrimiento de peluche. Además, con los siguientes dispositivos luminosos, se puede dotar al robot de una gran expresividad:

- 2 Pantallas uOLED de 128x128 píxeles que simulan los ojos, por los cuales se reproducen diferentes animaciones para simular las distintas emociones del robot, como puede ser enfado, tristeza, felicidad, sueño, entre otros, y el parpadeo de los ojos.

- 2 leds RGB en la cara del robot para simular las mejillas.

- 1 led RGB en el torso para simular el corazón. Este led cambia de color dependiendo del estado de ánimo del robot y simula los latidos encendiéndose y apagándose.

-1 led que simula la boca, el cual se enciende y se apaga mientras habla el robot, y dependiendo de la intensidad con la que hable.

Por otra parte, posee tres sensores de tacto situados en los hombros y tripa, controlados con una placa Arduino Mega. Para la percepción del entorno cuenta con una cámara Kinect, de la cual hablamos en la Sección 2.5.1, para detectar si alguien se aproxima, además de utilizar un micrófono en la parte media del robot. Mediante los sensores de tacto y/o el micrófono, se puede realizar la interacción con Mini. El reconocimiento de voz y el sintetizador se realiza mediante Loquendo [18]. Por último, para mostrar contenido multimedia, Mini cuenta con una Tablet, con la que además se pueden hacer otro tipo de actividades, como acceder a internet o realizar llamadas vía Skype.

A nivel de software, Mini trabaja sobre el framework ROS. Su control se basa en una máquina de estados programada con Smach [19], y mediante el sistema de diálogo IWAKI se controlan las transiciones. Posee dos paquetes denominados multimodal fussion y multimodal fission para controlar las entradas y salidas de este sistema de diálogo.

2.5.3 Framework ROS

Robot Operating System (ROS) es un sistema operativo de código abierto creado para el desarrollo de software del ámbito de la robótica. Se desarrolló originalmente en 2007 bajo el nombre de switchyard por el laboratorio de inteligencia artificial de Stanford a fin de dar soporte al proyecto del Robot con Inteligencia Artificial de Stanford. Nos otorga abstracción del hardware, control de los dispositivos a bajo nivel, implementación de las funcionalidades más comúnmente usadas, intercambio de mensajes entre procesos y manejo de paquetes. Además, nos da las herramientas y librerías para obtener, buildear, escribir y arrancar código entre varios computadores.

Está basado en una arquitectura de grafos donde el procesamiento toma lugar en los nodos que pueden recibir, mandar y multiplexar mensajes de sensores, control, estados, planificaciones y actuadores, entre otros. Implementa varios tipos diferentes de comunicación, incluyendo el estilo RPC de comunicación síncrono entre servicios, transmisión asíncrona de datos entre tópicos, y almacenamiento de datos en un servidor de parámetros. ROS no es un framework en tiempo real, pero es posible integrarlo con código en tiempo real. Las librerías están orientadas para un sistema UNIX, aunque también se están adaptando a otros sistemas operativos, aunque están en estado experimental.

ROS es un framework de procesos distribuido, llamados nodos, que permite que los ejecutables sean individualmente designados y aproximadamente acoplados durante el tiempo de ejecución. Estos procesos pueden agruparse dentro de paquetes, los cuales pueden ser fácilmente compartidos y distribuidos. ROS también soporta un sistema federado de código (Repositorios) que permite la colaboración entre usuarios para la distribución de código. Este diseño, desde el nivel de sistema de archivos hasta el nivel de comunidad, permite decisiones independientes sobre el desarrollo e implementación, pero todos pueden trabajar juntos con las herramientas de infraestructuras de ROS [20].

El framework de ROS puede ser implementado en cualquier lenguaje de programación moderno. Actualmente, está implementado en Python, C++ y Lisp, y existen librerías experimentales en Java y Lua.

A continuación, se resumen algunas de las funcionalidades que nos proporciona ROS y que se utilizarán a lo largo de este documento:

- **Mensaje**: estructuras de datos utilizados para la comunicación interna entre los nodos.

- **Tópico**: asunto donde se encuentran los mensajes. Un nodo puede suscribirse o publicar en dichos tópicos, es decir, puede tomar o meter mensajes en dichos tópicos.

- **Nodo**: programa ejecutable que realiza la función para la que ha sido programado. Es lo que define la funcionalidad de un sistema.

- **Archivo .bag**: archivo que guarda mensajes de determinados tópicos que puede reproducirse para ser leído por uno o varios nodos.

2.5.4 Scikit-Learn

Scikit-Learn es un software gratuito de librerías de Aprendizaje Automático para el lenguaje de programación Python [14]. Otorga varios algoritmos de clasificación, regresión y agrupación incluyendo Support Vector Machine, RandomForest, Gradient Boosting, k-means y DBSCAN, y está diseñado para interoperar con las librerías numéricas y científicas Numpy y Scipy.

Scikit-Learn proporciona una serie de algoritmos de aprendizaje supervisado y no supervisado por medio de una interfaz consistente en Python. Es distribuida bajo muchas distribuciones de Linux, dándole un uso tanto académico como comercial. Se le ha dado forma de biblioteca ya que da un nivel de robustez y el apoyo requerido para su uso en sistemas de producción. Esto significa que tiene un interés especial en que sea fácil de usar, que la calidad del código sea buena y en la colaboración, documentación y su desarrollo.

Algunos de los grupos de modelos dados por scikit-learn incluyen:

- Clustering (Agrupación): para agrupar datos no etiquetados. Un ejemplo sería K-Means.

- Cross Validation: para estimar el rendimiento de los modelos supervisados en datos no vistos.

- Datasets: para testear datasets y para generar datasets con propiedades específicas para estudiar el comportamiento de los modelos.

- Reducción de la dimensionalidad: para reducir el número de atributos en los datos por resumen, visualización, y selección de características. Un ejemplo sería: Principal component analysis (PCA).

- Métodos de ensamblaje: para combinar las predicciones de múltiples modelos supervisados.

- Extracción de características: para definir atributos en datos de texto o imágenes.

- Selección de características: para identificar atributos significativos de los cuales crear modelos supervisados.

- Modificación de parámetros: para obtener el mayor rendimiento de los modelos supervisados.

- Aprendizaje múltiple: para resumir y demostrar datos complejos multidimensionales.

- Modelos supervisados: un amplio array no limita modelos lineales generalizados, análisis discriminatorios, naive bayes, métodos lazy, redes neuronales, support vector machines y decision trees.

Existen, además de Scikit-Learn, otros programas que nos podrían dar las herramientas necesarias para utilizar el Aprendizaje Automático en el proyecto. Algunos ejemplos serían: Shogun [21], Weka [22] y MLlib[2], pero nos hemos decantado finalmente por Scikit-Learn debido a que es compatible con Python y es fácil de utilizar e implementar en el código ya que viene en forma de librerías fácilmente exportables.

2.5.5 Sistemas de detección facial

Se puede definir la detección facial por medio de un computador como la ubicación de los rostros que se encuentran presentes en una imagen o un video. Esta detección no es tan sencilla como la que realizamos los humanos, puesto que la realizamos de forma instantánea. Esto es debido al gran paralelismo que existe en las redes neuronales del cerebro. Aun así, existen distintos métodos para realizar esta detección por medio de una computadora [23]:

- Métodos basados en conocimiento: se codifica la manera de conocer lo que es un rostro. Normalmente son atributos geométricos codificados en forma de reglas.

[2] Página oficial de MLlib: https://spark.apache.org/mllib/ (último acceso el 10/09/2016)

- Métodos basados en características invariantes: por medio de la información del color y la textura de la imagen, se puede representar los rostros sin tener en cuenta su orientación.

- Métodos basados en plantillas: se puede considerar una técnica basada en el conocimiento. Se detecta el rostro empleando una familia de curvas que representan el objeto.

- Métodos basados en la apariencia: para esta técnica se requiere de varias imágenes. A partir de ellas, el sistema aprende y se consigue codificar solamente lo necesario para hacer la detección de las características de interés del rostro humano.

Para realizar la detección facial del presente proyecto, emplearemos la librería Stasm, la cual es una librería en C++ e implementada para OpenCV. A partir de esto, somos capaces de encontrar los puntos característicos en los rostros. Este sistema recibe como entrada una imagen del rostro y devuelve la posición de los puntos que lo definen. Este software está diseñado para trabajar con vistas frontales de caras y expresiones neutrales [24]. En la Figura 2.7 podemos ver cómo actúa el software Stasm. Para el presente trabajo, se empleará dicho software para caracterizar el rostro del usuario. A partir de los puntos obtenidos, filtraremos los resultados para obtener los 18 puntos que delimita la boca. Con esos puntos podremos continuar con el desarrollo del proyecto. Todo este proceso es una parte heredada del proyecto "Diseño e implementación de un sistema de reconocimiento vocal a través de información RGB-D".

Figura 2.7: Funcionamiento de Stasm

Este sistema está basado en Active Shape Models (modelos de forma activa) [25]. Esto se define como modelos paramétricos deformables donde un modelo estadístico de la variación global de la forma del objeto es generado a partir de un conjunto de entrenamiento consistente en imágenes anotadas. Este modelo, conocido como Modelo de Distribución de Puntos, es utilizado posteriormente para ajustar una plantilla a instancias del objeto no presentes en el conjunto de entrenamiento. La forma del objeto es representada como un conjunto de puntos.

3. Análisis y diseño del sistema

En este capítulo se describirá el análisis y el diseño del sistema a desarrollar. Hablaremos de las recomendaciones que se deben seguir en el sistema, y el entorno operacional del mismo. Tras esto, detallaremos los distintos tipos de requisitos que debe cumplir el sistema junto con los casos de uso de cada uno.

3.1 Restricciones del proyecto

En esta sección de mostrarán las normas y restricciones que debe de cumplir el sistema. Las restricciones que se deben seguir se dividirán en restricciones de hardware y software:

- **Restricciones de hardware:**
    - El sensor para captar imágenes de usuario debe de ser la Kinect XBOX 360.
    - El robot utilizado para la terapia debe de ser el robot Mini perteneciente al grupo de investigación RoboticsLAb de la UC3M.
    - El sensor Kinect debe de estar conectado al robot, el cual gestionará la transmisión de datos al sistema.
- **Restricciones de software:**
    - El ordenador que se utilice debe de tener instalado ROS Índigo.
    - Se debe tener instalado un entorno de programación de Python para ejecutar el sistema.
    - El ordenador que se utilice debe de tener instalado el sistema operativo Ubuntu 14.04.
    - Se debe instalar el software Scikit-Learn para la implementación de los clasificadores necesarios para el funcionamiento del sistema.

- Se debe instalar el software Stasm en el sistema para la detección de rostros.

3.2 Entorno operacional

En este apartado definiremos el entorno operacional del presente proyecto. Se determinará las herramientas de software y hardware empleadas durante el desarrollo del proyecto:

- Se debe tener una cámara Kinect XBOX 360 conectada al ordenador.
- Ordenador con el sistema operativo Ubuntu 14.04.
- Librerías Scikit-Learn.
- Software Stasm.
- Paquete Microsoft Office.
- Ubuntu Linux 14.04.
- Entorno de programación Python.
- Editor de textos Jupyter Notebook.
- Framework ROS

3.3 Especificación de requisitos

En la siguiente sección describiremos los tipos de requisitos que se debe de tener en cuenta en el proyecto. Definiremos los requisitos utilizando el modelo de la Tabla 3.1.

Tabla 3.1: Modelo de descripción de requisitos

REQUISITO
Descripción:
Prioridad:
Necesidad:
Claridad:
Verificabilidad:

Debemos definir cada punto de la tabla para dejar claro la funcionalidad de la misma:

- **Descripción**: se dará una breve explicación del requisito.

- **Prioridad**: expresa la relevancia que tiene el requisito frente al resto para facilitar la organización del proyecto.

    - Alta: el requisito tiene la mayor prioridad posible.
    - Media: el requisito tiene menor prioridad frente a los de prioridad alta, pero sigue tiene prioridad frente algunos requisitos.
    - Baja: el requisito tiene la menor prioridad posible.

- **Necesidad**: expresa lo imprescindible que es el requisito para el desarrollo del proyecto. Puede tomar tres niveles:

    - Esencial: el requisito es completamente necesario para la resolución del proyecto.
    - Deseable: el requisito es importante para el desarrollo del proyecto, pero si no se consigue, no compromete la resolución del proyecto
    - Opcional: el requisito puede llegar a cumplirse, pero el no cumplimiento no compromete ninguna parte del proyecto.

- **Claridad**: expresa el nivel de interpretación que debe tener el requisito. Puede tomar tres niveles:

    - Alta: con este nivel se indica que el requisito debe de tener una única interpretación. No se aceptan ambigüedades.
    - Media: el requisito puede tener pequeñas ambigüedades.
    - Baja: el requisito tendrá varias ambigüedades, por lo que se le podrá dar bastantes interpretaciones.

- **Verificabilidad**: expresa la posibilidad de poder comprobar si se ha añadido el requisito en el proyecto:

    - Alta: el requisito se puede comprobar fácilmente.
    - Media: se debe realizar una comprobación para confirmar la existencia del requisito.
    - Baja: no hay posibilidad de comprobar la existencia del requisito.

3.3.1 Requisitos funcionales

En este capítulo se describirán los requisitos funcionales que debe cumplir el presente proyecto. Estos determinan los comportamientos, funcionalidades y objetivos específicos que el sistema debe cumplir. En las Tablas de la 3.2 a la 3.14 podemos ver los requisitos funcionales del presente proyecto:

Tabla 3.2: Requisito funcional 1

RF-1: Reconocimiento facial
Descripción: el software debe ser capaz de reconocer el rostro de un único usuario.
Prioridad: Alta
Necesidad: Esencial
Claridad: Alta
Verificabilidad: Alta

Tabla 3.3: Requisito funcional 2

RF-2: Identificación de la boca
Descripción: el sistema debe obtener los 18 con los que el software caracteriza la boca del usuario
Prioridad: Alta
Necesidad: Esencial
Claridad: Alta
Verificabilidad: Alta

Tabla 3.4: Requisito funcional 3

RF-3: Normalización
Descripción: el sistema debe normalizar los 18 puntos que se obtienen del rostro.
Prioridad: Media
Necesidad: Opcional
Claridad: Alta
Verificabilidad: Media

Tabla 3.5: Requisito funcional 4

RF-4: Generación de ficheros para entrenar los clasificadores
Descripción: el sistema debe crear los ficheros donde guarde la información de los datos obtenidos por la Kinect para el posterior entrenamiento de los clasificadores.
Prioridad: Alta
Necesidad: Esencial
Claridad: Alta
Verificabilidad: Alta

Tabla 3.6: Requisito funcional 5

RF-5: Obtención de los mensajes recibidos por la detección
Descripción: el sistema debe recibir los mensajes recibidos por el nodo de detección.
Prioridad: Alta
Necesidad: Esencial
Claridad: Alta
Verificabilidad: Media

Tabla 3.7: Requisito funcional 6

RF-6: Entrenamiento del clasificador
Descripción: a partir de los datos de los ficheros guardados, se entrena el clasificador.
Prioridad: Alta
Necesidad: Esencial
Claridad: Alta
Verificabilidad: Media

Tabla 3.8: Requisito funcional 7

RF-7: Generación del fichero para guardar el clasificador
Descripción: el sistema debe crear el fichero que guarde la configuración del clasificador entrenado.
Prioridad: Alta
Necesidad: Esencial
Claridad: Alta
Verificabilidad: Alta

Tabla 3.9: Requisito funcional 8

RF-8: Obtención del clasificador entrenado
Descripción: se debe ser capaz de leer los archivos que poseen la configuración del clasificador
Prioridad: Alta
Necesidad: Esencial
Claridad: Alta
Verificabilidad: Media

Tabla 3.10: Requisito funcional 9

RF-9: Resolución de la clasificación
Descripción: el clasificador debe de dar un resultado conforme a los datos que le lleguen de la detección
Prioridad: Media
Necesidad: Esencial
Claridad: Alta
Verificabilidad: Alta

Tabla 3.11: Requisito funcional 10

RF-10: Gestión de los resultados
Descripción: el sistema deberá estudiar durante un determinado tiempo los resultados del clasificador para determinar el resultado final.
Prioridad: Media
Necesidad: Esencial
Claridad: Media
Verificabilidad: Media

Tabla 3.12: Requisito funcional 11

RF-11: Envío de mensajes al robot
Descripción: el sistema debe mandar la información, por medio de mensajes, correspondiente a cómo debe actuar el robot frente a los resultados obtenidos.
Prioridad: Media
Necesidad: Esencial
Claridad: Alta
Verificabilidad: Alta

Tabla 3.13: Requisito funcional 12

RF-12: Reproducción de audio
Descripción: el robot debe ser capaz de reproducir los audios correspondientes a la respuesta a los resultados del clasificador
Prioridad: Media
Necesidad: Deseable
Claridad: Alta
Verificabilidad: Alta

Tabla 3.14: Requisito funcional 13

RF-13: Visualización de imágenes
Descripción: el robot debe de ser capaz de transmitir las imágenes correspondientes a los resultados dados por el sistema
Prioridad: Media
Necesidad: Deseable
Claridad: Alta
Verificabilidad: Alta

3.3.2 Requisitos no funcionales

En la siguiente sección se describirán los requisitos no funcionales utilizando el modelo presentado anteriormente. Los requisitos no funcionales pueden definirse como las restricciones a las características del diseño del presente proyecto. En las Tablas de la 3.15 a la 3.23 se pueden observar los requisitos no funcionales del presente proyecto:

Tabla 3.15: Requisito no funcional 1

RNF-1: Compatibilidad del sistema con el robot Mini
Descripción: el sistema debe de ser capaz de mandar y recibir mensajes con el robot Mini.
Prioridad: Alta
Necesidad: Esencial
Claridad: Alta
Verificabilidad: Alta

Tabla 3.16: Requisito no funcional 2

RNF-2: Compatibilidad con el framework ROS Indigo
Descripción: el sistema debe ser capaz de trabajar con la versión índigo de ROS, ya que es con la que funciona el robot Mini.
Prioridad: Alta
Necesidad: Esencial
Claridad: Alta
Verificabilidad: Media

Tabla 3.17: Requisito no funcional 3

RNF-3: Compatibilidad con Ubuntu 14.04
Descripción: el sistema debe de ser compatible con el sistema operativo Ubuntu 14.04.
Prioridad: Alta
Necesidad: Esencial
Claridad: Alta
Verificabilidad: Media

Tabla 3.18: Requisito no funcional 4

RNF-4: Empleo del lenguaje Python
Descripción: el software utilizado en el sistema debe de estar escrito en lenguaje Python
Prioridad: Alta
Necesidad: Esencial
Claridad: Alta
Verificabilidad: Media

Tabla 3.19: Requisito no funcional 5

RNF-5: Uso de las librerías Scikit-Learn
Descripción: el sistema debe de utilizar las librerías Scikit-Learn para el desarrollo del proyecto
Prioridad: Alta
Necesidad: Esencial
Claridad: Alta
Verificabilidad: Alta

Tabla 3.20: Requisito no funcional 6

RNF-6: Obtención de buenos resultados en la clasificación
Descripción: los resultados obtenidos por el clasificador del sistema deben de ser aceptables.
Prioridad: Alta
Necesidad: Esencial
Claridad: Alta
Verificabilidad: Alta

Tabla 3.21: Requisito no funcional 7

RNF-7: Interfaz intuitiva
Descripción: el sistema debe de ser fácilmente manejable por parte de usuarios ajenos al diseño
Prioridad: Media
Necesidad: Deseable
Claridad: Alta
Verificabilidad: Alta

Tabla 3.22: Requisito no funcional 8

RNF-8: Interacción usuario-robot
Descripción: el robot debe de tener una interacción correcta con el paciente para apoyar la terapia.
Prioridad: Media
Necesidad: Deseable
Claridad: Alta
Verificabilidad: Alta

Tabla 3.23: Requisito no funcional 9

RNF-9: Gestión de ficheros
Descripción: el sistema debe de ser capaz de crear, modificar y buscar ficheros.
Prioridad: Alta
Necesidad: Esencial
Claridad: Alta
Verificabilidad: Alta

3.4 Casos de uso

3.4.1 Descripción gráfica

En la Figura 3.1 se detallará el diagrama de los casos de uso:

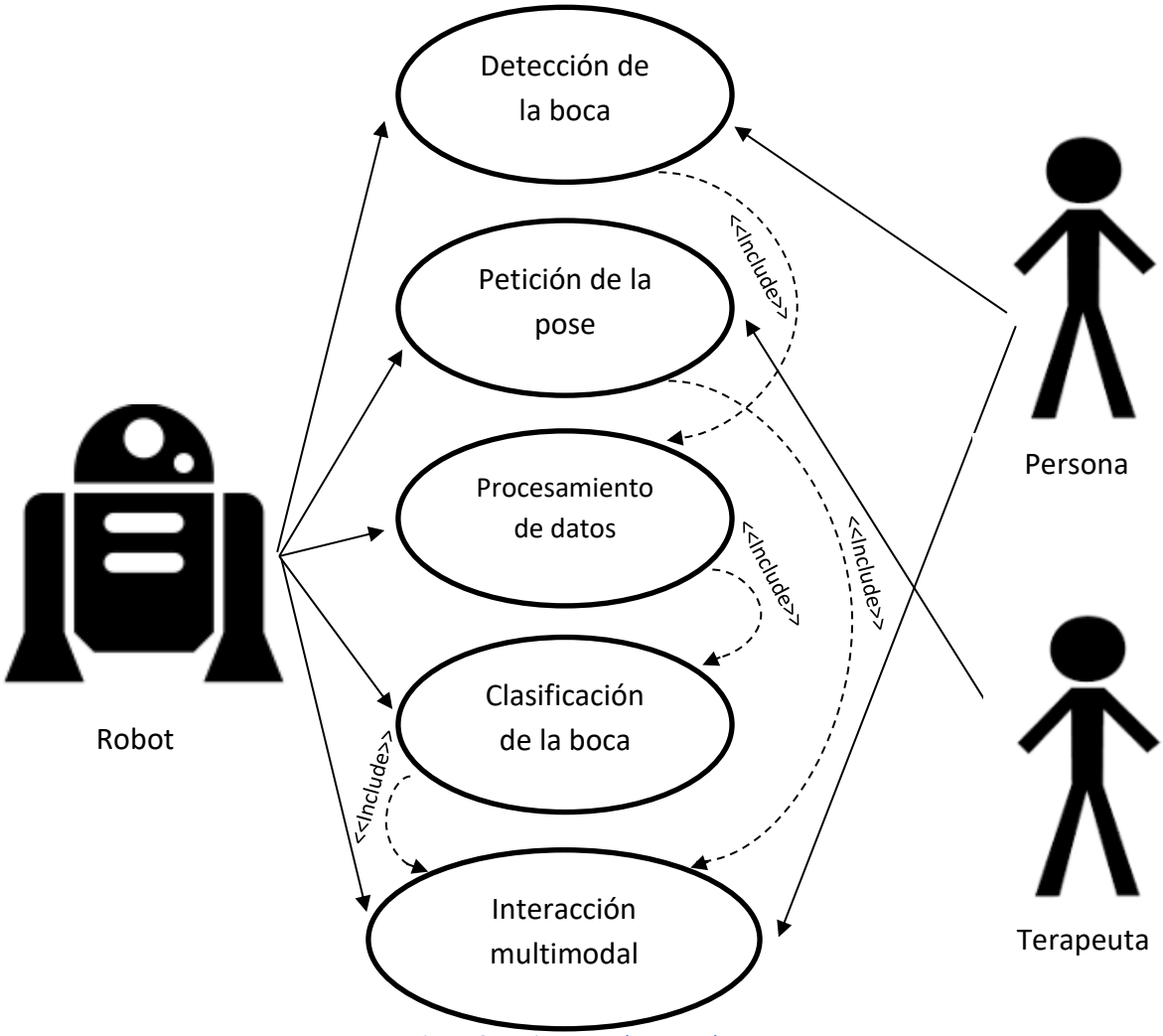


Figura 3.1: Diagrama de casos de uso

3.4.2 Descripción tabular

En el siguiente capítulo procederemos a definir los casos de uso del sistema, los cuales podemos ver en las Tablas de la 3.25 a la 3.29. Los casos de uso son las actuaciones que debe de tener un actor para realizar una tarea concreta con el sistema. Para definir los casos de uso utilizaremos el modelo presentado en la Tabla 3.24:

Tabla 3.24: Ejemplo de caso de uso

CASO DE USO
Nombre:
Actor:
Descripción:
Precondiciones:
Postcondiciones:

En primer lugar, definiremos los actores que intervendrán en los casos de uso:

- **Robot**: este elemento hace referencia a la comunicación de mensajes entre el robot Mini y el sistema. Se pueden captar los mensajes que manda el sensor Kinect al sistema y los mensajes que manda el sistema para que actúe el robot.
- **Usuario**: este elemento hace referencia a la persona que interactúa con el robot. Esta es la encargada de pronunciar las poses vocales frente a la cámara Kinect.
- **Terapeuta**: este elemento hace referencia a la persona encargada de supervisar la terapia.

A continuación, definiremos los elementos más relevantes de dicha tabla:

- **Descripción**: se definirá el objetivo del caso de uso.
- **Dependencias**: se enumerarán los casos de usos previos necesarios para realización del caso de uso
- **Precondiciones**: se definirán las condiciones iniciales para que se pueda realizar el caso de uso.
- **Secuencia**: se enumerarán los pasos necesarios para la realización del caso de uso.
- **Postcondiciones**: se definirán las consecuencias producidas por la realización del caso de uso.

Tabla 3.25: Caso de uso 1

CU-1
Nombre: Detección boca.
Actor: Robot, usuario.
Dependencias: no tiene
Descripción: el robot captura las imágenes por medio del sistema Kinect y detecta la boca del usuario.
Precondiciones: ▪ **El robot y el sistema deben de estar en ejecución.** ▪ **El usuario debe de estar frente al robot.**
Secuencia: ▪ **Se conecta el sensor Kinect al sistema** ▪ **El usuario coloca su rostro frente al sensor** ▪ **Se inicializa la toma de datos por medio del sensor Kinect** ▪ **Se inicia el nodo Detection**
Postcondiciones: ▪ **El robot devuelve los 18 puntos correspondientes a la caracterización de la boca.**

Tabla 3.26: Caso de uso 2

CU-2
Nombre: Petición de la pose.
Actor: Robot, terapeuta.
Dependencias: no tiene
Descripción: el terapeuta indica al robot la pose vocal que va a vocalizar para la realización de la terapia.
Precondiciones: ▪ **El robot y el sistema deben de estar en ejecución.** ▪ **La interfaz debe de estar activada.**
Secuencia: ▪ **El sistema pregunta al usuario qué pose pretende pronunciar.** ▪ **El terapeuta introduce por teclado la pose deseado.** ▪ **El sistema guarda la pose para su posterior utilización.**
Postcondiciones: ▪ **Se obtienen los datos de la pose a detectar.**

Tabla 3.27: Caso de uso 3

CU-3
Nombre: Procesamiento de los datos obtenidos.
Actor: Robot.
Dependencia: ▪ **CU-1: Detección boca.**
Descripción: el robot manipula los datos obtenidos por el sensor Kinect para su posterior uso en el clasificador.
Precondiciones: ▪ **El robot y el sistema deben de estar en ejecución.** ▪ **El sensor Kinect debe detectar correctamente.** ▪ **El nodo Detección debe estar encendido y debe enviar los datos de la boca correctamente.**
Secuencia: ▪ **Se toman los puntos provenientes del nodo Detection.** ▪ **Se normalizan los puntos obtenidos.** ▪ **Se guardan los puntos normalizados para después clasificarlos.**
Postcondiciones: ▪ **Los datos obtenidos por el sensor Kinect son normalizados y están listos para su clasificación.**

Tabla 3.28: Caso de uso 4

CU-4
Nombre: Clasificación de la boca.
Actor: Robot
Dependencias: ▪ **CU-3: Procesamiento de los datos obtenidos**
Descripción: el robot clasifica la pose que ha detectado.
Precondiciones: ▪ **El robot y el sistema deben de estar en ejecución.** ▪ **El clasificador debe de estar entrenado correctamente.** ▪ **Los datos de la boca deben de estar correctamente manipulados.**
Secuencia: ▪ **Se obtienen los 18 puntos normalizados que conforman la boca del usuario.** ▪ **Se introducen los puntos en el clasificador.** ▪ **Por cada pose, se obtiene la probabilidad de que el usuario la esté pronunciando.** ▪ **A partir de las probabilidades obtenidas, se estima qué pose vocal es la realizado.**
Postcondiciones: ▪ **Se obtiene la clasificación de la pose dicha por el usuario.** ▪ **Se puede comprobar si coincide con la pose pedida.**

Tabla 3.29: Caso de uso 5

CU-5
Nombre: Interacción multimodal.
Actor: Robot, usuario.
Dependencias: ▪ **CU-2: Petición de la pose.** ▪ **CU-4: Clasificación de la boca.**
Descripción: el robot, dependiendo si el resultado es correcto o incorrecto, felicita al usuario o le da instrucciones para corregir la pose de la boca.
Precondiciones: ▪ **El robot y el sistema deben de estar en ejecución.** ▪ **El usuario debe de estar frente al robot.** ▪ **El sensor Kinect debe de estar conectado.** ▪ **Debe de llegar al sistema la pose vocal obtenida por la clasificación previa.**
Secuencia: ▪ **Se toman los resultados de la clasificación.** ▪ **A partir de los resultados, se decide que indicación dar al usuario.** ▪ **Se envía mediante la síntesis de voz del robot las indicaciones pertinentes, ya sea una corrección o una felicitación.**
Postcondiciones: ▪ **En el caso de que el resultado sea correcto, se felicitará al usuario y se le propondrá una nueva pose.** ▪ **En el caso de que el resultado sea incorrecto, se le dará al usuario las correcciones necesarias para corregir dicho fallo.**

3.5 Metodología

En la siguiente sección nos encargaremos de definir la metodología. La que se ha usado para el desarrollo del trabajo está basada en prototipos, y la explicaremos más adelante. Para la organización de tareas se ha usado una metodología de gestión en cascada.

Un prototipo se puede definir como un diseño rápido de una parte del sistema o del sistema entero, donde se quieren desarrollar determinadas partes del sistema para que puedan ser vistas por un usuario. Mediante la evaluación del sistema por parte del usuario, se consigue mejorar el sistema y su funcionamiento. Hay que tener en cuenta que el prototipo debe mejorarse para aumentar su calidad frente al usuario, ya que sigue considerándose un diseño rápido del sistema.

Se ha decidido emplear este tipo de metodología debido a que es la más indicada para la programación de este proyecto. Se busca el cumplimiento de determinados hitos en cada prototipo para alcanzar finalmente el sistema final. Con esto, además, se busca mejorar progresivamente el alcance del software, añadiéndole utilidades u optimizando las que ya posee.

4. Implementación del sistema

A continuación, se describirá el proceso de implementación del sistema. Se comenzará detallando la integración de las tecnologías, donde hablaremos de las funciones y las clases creadas para el desarrollo del sistema. Se estudiarán las funciones más importantes del proyecto, y se finalizará explicando el dominio utilizado.

4.1 Integración de los componentes

Podemos definir el objetivo final del presente trabajo como la creación de dos nodos para integrar en el robot Mini utilizando la arquitectura propia de este robot. Se podría migrar dicho sistema a otro robot, siempre que acepte el sistema de mensajes que emplea el framework ROS. En la Figura 4.1, podemos observar la arquitectura del proyecto:

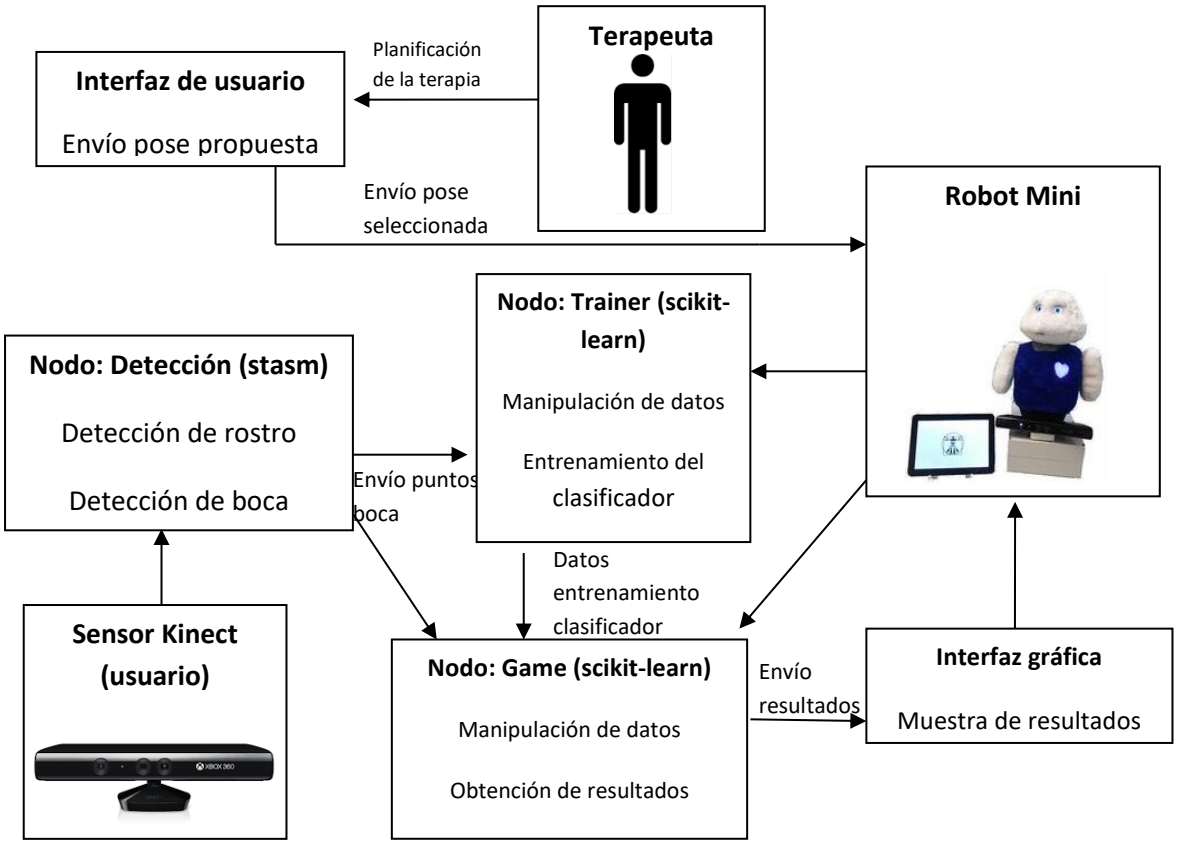


Figura 4.1: Arquitectura del sistema

4.1.1 Integración ROS

Como se indicó la Sección 2.5.3, se emplea el framework ROS para el envío de mensajes entre el sistema y el robot, además de incluir la comunicación entre los nodos del sistema [20]. Debido a esto, se ha decidido instalar este framework en la arquitectura de nuestro sistema. Además, este sistema nos permite lanzar instrucciones al robot ya sea en lenguaje C++ o Python. En nuestro proyecto emplearemos tanto el lenguaje C++ como el lenguaje Python. Esto es debido a que se utiliza un nodo del proyecto previo, ya escrito en C++, y librerías escritas en Python en otros nodos, lo que nos fuerza a usar dicho lenguaje.

Gracias a ROS, podremos enviar toda la información correspondiente a los puntos que determinan la boca del usuario de un nodo a otro, y podremos enviar las reacciones finales al robot que dependerán de los resultados del clasificador. Para las reacciones enviaremos al robot los archivos de texto de lo que este debe decir, como imágenes que debe visualizar en su Tablet.

En el comienzo del proyecto, se desconocía el uso del framework ROS. Para documentarnos acerca de él, se siguieron los tutoriales ofrecidos por la página oficial de ROS[3]. En estos tutoriales se aprendieron las nociones básicas de dicho framework, la creación y manejo de paquetes y nodos y el uso de mensajes. También se aprendieron las nociones de publicador y suscriptor, y como programarlos. Gracias a estos tutoriales se aprendió a movernos dentro de este framework, además de sus comandos, que nos ofrecen determinadas funcionalidades, lo que nos permitió proseguir con el desarrollo del proyecto.

Una vez hemos adquirido el conocimiento acerca de ROS, llevaremos a cabo la creación de los nodos que determinan el presente proyecto. Tendremos las capacidades de escribir en cualquier lenguaje los publicadores y los suscriptores necesarios para el envío de información en nuestro sistema, y las nociones para crear el paquete que contendrá los nodos creados. Además, se sabrá cómo utilizar bags donde guardaremos la información necesaria para el entrenamiento del clasificador, lo que facilitará mucho el desarrollo del proyecto.

Podemos concluir con que el estudio de ROS y la aplicación de los conocimientos obtenidos han sido satisfactorios. Ha sido una de las partes del trabajo que más tiempo nos ha llevado debido al desconocimiento total que se tenía al comienzo del presente proyecto.

[3] Tutoriales de ROS: http://wiki.ros.org/ROS/Tutorials (último acceso el 10/09/2016)

4.1.2 Integración Scikit-Learn

Scikit-Learn es una librería gratuita software de Machine Learning que empleamos en el presente proyecto. Estas librerías están escritas en Python y se originaron como un proyecto de David Cournapeau para el proyecto "Google Summer of Code", donde se crea una herramienta para Scipy, de ahí el nombre de Sci-Kit. Scikit-Learn nos da los recursos necesarios para poder crear nuestro clasificador correctamente. Nos ofrece una amplia gama de clasificadores para poder elegir el más ajustado a nuestras necesidades, además de poder ajustar sus parámetros de forma sencilla para una mejor eficiencia. Por lo tanto, se propuso la integración de estas librerías en el sistema para poder realizar un correcto uso de las técnicas de Machine Learning y de los clasificadores que lo hacen posible.

Para llevar a cabo la integración de dicho software, se utilizó la guía de instalación que nos ofrece la página web oficial de Scikit-Learn[4]. Con ellas, instalamos todas las librerías de Scikit-Learn a través del repositorio Anaconda, lo que hace posible el uso de funciones y clases de esta librería en los nodos que creemos en nuestro sistema [26].

Una vez instalado el software, inicialmente no se tenía ningún tipo de conocimiento sobre su uso. La página web de Scikit-Learn[5] nos ofrece unos tutoriales donde nos enseña las nociones básicas del Aprendizaje Automático, que tipo de aprendizaje debemos utilizar, que en este caso será aprendizaje supervisado basado en clasificación, y cómo utilizar las diferentes funciones y clases que nos ofrecen las librerías. Además, se nos ofrece una explicación teórica de cada clasificador, la manera correcta de utilizarlo, y en que contextos es más favorable su implementación. Incluso nos ofrecen ejemplos prácticos donde se muestra el uso de dichos clasificadores, lo que nos ayuda en gran medida a usar las funciones de un clasificador específico de forma correcta.

Una vez adquiridos los conocimientos necesarios para utilizar el software Scikit-Learn, podemos comenzar a utilizarlo en nuestro sistema. Para comprobar la eficacia de cada clasificador, debemos crear un prototipo con cada uno y anotar los resultados que nos da para comparar y elegir el clasificador que mejor responde en este caso. Con esto, podremos utilizar el clasificador con unos resultados aceptables y satisfactorios, lo que nos permite seguir adelante y progresando con nuestro sistema y así alcanzar los objetivos propuestos.

[4] Guia de instalación de Scikit-Learn: http://scikit-learn.org/stable/install.html (último acceso el 10/09/2016)
[5] Tutoriales de Scikit-Learn: http://scikit-learn.org/stable/tutorial/index.html (último acceso el 10/09/2016)

4.1.3 Integración Stasm

El software Stasm es una librería para OpenCV escrita en C++ utilizada para la detección de rostros [25]. Este software se encarga de caracterizar el rostro del usuario y devolvernos los puntos en el espacio donde se encuentra, lo que nos permite trabajar con la posición de la cara del usuario. Se utiliza en el nodo heredado del proyecto previo en el cual se logra caracterizar los 18 puntos de la boca de la persona que esté en frente de la Kinect. Al ser una parte de un proyecto previo, no ha sido modificado el código del nodo heredado, pero si se ha tenido que proceder a la integración de dicho software para el correcto funcionamiento del sistema.

Para realizar la integración, se han seguido los tutoriales encontrados en la siguiente página web[6], donde se nos indica cómo descargarnos todo el software necesario y como ajustarlo todo para que pueda funcionar en nuestro sistema. En el Apéndice A.3 se describe detalladamente cómo realizar la instalación de Stasm. Tras realizar todos estos pasos, ya seremos capaces de utilizar el nodo Detección sin ningún problema, lo que nos dará la información necesaria para proseguir nuestro trabajo.

4.1.4 Herramientas para el desarrollo del componente

Durante el transcurso del presente proyecto, nos hemos ayudado de diferentes programas que nos han facilitado su desarrollo. Estas herramientas son tanto utilidades dadas por el framework ROS como programas externos. Nos han proporcionado determinadas funcionalidades que han hecho posible la creación de este sistema. Estas herramientas son:

- **Jupyter Notebook**

Jupyter Notebook[7] es una aplicación servidor-cliente que permite editar y probar documentos por medio del navegador web [27]. Los archivos notebook son los que poseen tanto lenguaje de programación (Python) como elementos de texto enriquecido (ecuaciones, figuras, links). Estos documentos pueden ser leídos tanto por una persona conteniendo información como ser ejecutados para realizar análisis de datos. En la Figura 4.2 podemos ver un ejemplo de la apariencia de esta aplicación:

[6] Guia de instalación de Stasm: http://www.milbo.users.sonic.net/stasm/ (último acceso el 12/09/2016)
[7] Explicación de la funcionalidad de Jupyter Notebook: http://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#references (último acceso el 12/09/2016)

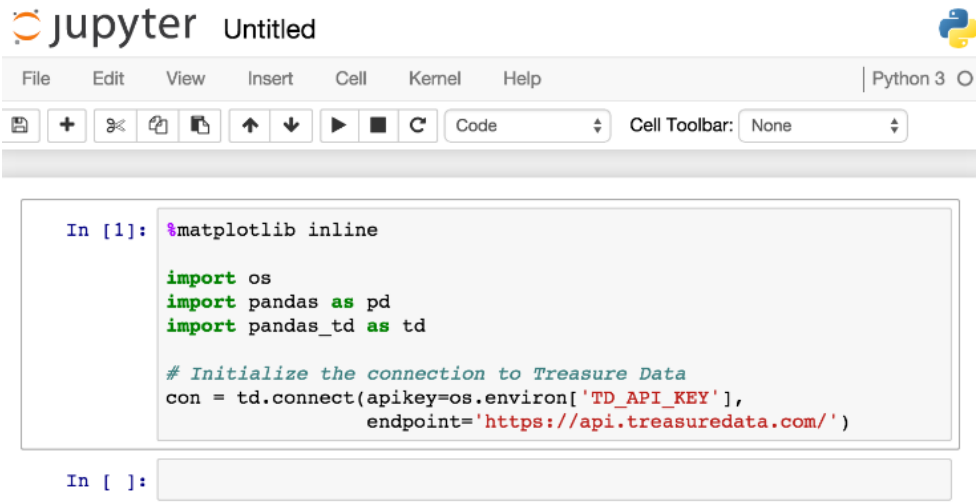


Figura 4.2: Interfaz de Jupyter Notebook

Esta herramienta ha sido utilizada para la escritura de los nodos que han conformado el sistema. Este software permite programar en lenguaje Python y corregir en la propia interfaz las líneas escritas. Esto nos permite detectar los errores cometidos y así poder arreglarlos fácilmente. Además, acepta distintos de documentos ya sean .txt, como .py, como los archivos notebook propios de esta aplicación. Se puede utilizar con el sistema operativo Ubuntu 14.04, lo que lo hace compatible con nuestro sistema, ya que este es el sistema operativo en el que debemos trabajar en el proyecto. Por todo esto, consideramos que este software es el adecuado para programar los nodos de nuestro sistema.

- **Rosbag**

Esta herramienta no es una aplicación en sí, es una utilidad dada por el framework ROS que nos facilita la tarea en nuestro trabajo. Un archivo .bag es un formato de archivo de ROS que se utiliza para guardar datos de mensajes de ROS. Estos archivos los creamos para guardar los datos que nos llegan de la Kinect, que una vez procesados por el nodo Detección podemos analizar y manipular posteriormente. Estos archivos .bag se pueden crear fácilmente con la herramienta rosbag[8], la cual se suscribe a determinados tópicos y guarda la información de ellos para poder reproducirlos en otro momento. Las funciones dentro de rosbag que hemos empleado para el manejo de bags en nuestro sistema son:

[8] Explicación del funcionamiento de Rosbag: http://wiki.ros.org/rosbag (último acceso el 12/09/2016)

- **Rosbag record**: esta herramienta permite guardar la información de determinados tópicos en un archivo .bag del cual definimos previamente su nombre. Con esto queremos guardar la información proveniente del sensor Kinect para su posterior manejo en los nodos del sistema. Para nuestro caso, el método de ejecución sería inicializar la captura de imágenes desde el sensor Kinect y guardar la información de los siguientes tópicos:

    - */camera/depth/image-raw*: en este tópico se guarda la información de la imagen en profundidad.
    - */camera/depth/points*: en este tópico se guardan los puntos de la imagen visualizada en 3D
    - */camera/rgb/image_color*: en este tópico se guarda la imagen 2D en color.

    Con esto, conseguimos guardar la información suficiente para hacer funcionar el resto de nodos con los datos correctos sobre la posición de la boca del usuario.

- **Rosbag play**: con esta herramienta podemos leer los datos previamente guardados con rosbag record. De esta manera, podemos obtener la información de los tópicos que hemos grabado y usar los datos en el nodo Detección. Esta herramienta nos da la opción de reproducir archivos .bag en bucle o únicamente una vez. Utilizando esta sentencia, podemos abrir la información de diferentes usuarios para el entrenamiento del clasificador, lo que nos ahorra tener que entrenar en directo, con los errores que eso conllevaría. En el Apéndice B.1 hay un ejemplo de cómo usar este comando.

➢ **Rviz**

Este software es una herramienta del framework ROS. Su nombre proviene de ROS visualization. Este software es un visualizador 3D que nos permite ver la información de los sensores conectados. En la Figura 4.3 podemos ver un ejemplo de la interfaz de Rviz[9]. Con este programa hemos podido visualizar la imagen del Kinect antes de iniciar la rutina rosbag record, lo que ha reducido en gran medida los errores producidos durante las grabaciones. Además, nos permite visualizar los datos que contienen los bags grabados, lo que nos permite asegurarnos que la calidad de las imágenes grabadas es buena.

[9] Explicación del funcionamiento de Rvjz: http://wiki.ros.org/rviz (último acceso el 12/09/2016)

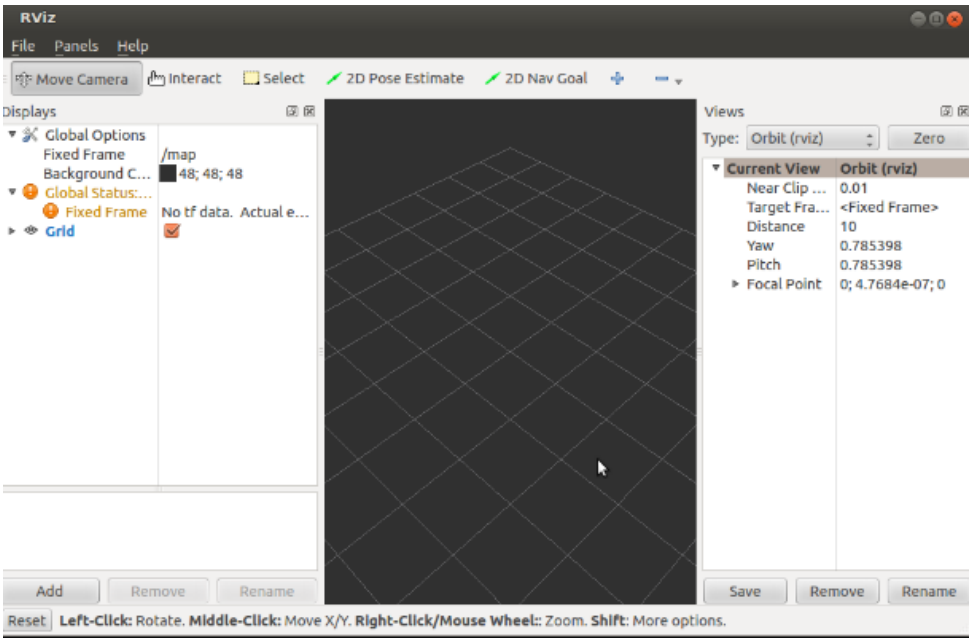


Figura 4.3: Interfaz de Rviz

4.2 Entrenamiento del clasificador

Una vez integrados e instalados todos los elementos que conforman el presente proyecto, podemos comenzar a realizar el entrenamiento del clasificador. Esta parte será programada en lenguaje Python, debido a que es el lenguaje que exigen las librerías de Scikit-Learn. En la Figura 4.4 se muestra la secuencia con la que se realiza la tarea del entrenamiento. A continuación, describiremos los elementos que conforman el diagrama de flujo.

1. **Inicio**: antes de comenzar cualquier acción dentro del sistema, debemos inicializar el flujo de datos contenidos dentro de los bags. Utilizando la sentencia rosbag play (ver Sección 4.1.4), seleccionamos el bag que queremos leer para entrenar el clasificador y arrancamos el nodo Detección para obtener los 18 puntos correspondientes a la boca.

2. **Cargar datos/Crear ficheros .txt nuevos**: en este punto se realiza una comprobación de si existen datos de entrenamiento previos**.** Si el sistema no encuentra ningún fichero de puntos de entrenamiento previo, se crean los ficheros correspondientes para poder guardar los puntos en un futuro. En el caso de que sí existan dichos archivos, se guardan los datos que contienen en una variable para su uso posterior.

3. **Preguntar pose**: se pregunta al usuario la pose vocal que se va a entrenar. El sistema comprueba si se ha introducido un valor correcto para continuar con el desarrollo del programa, ya que, en caso de error, se volverá a pedir una pose.

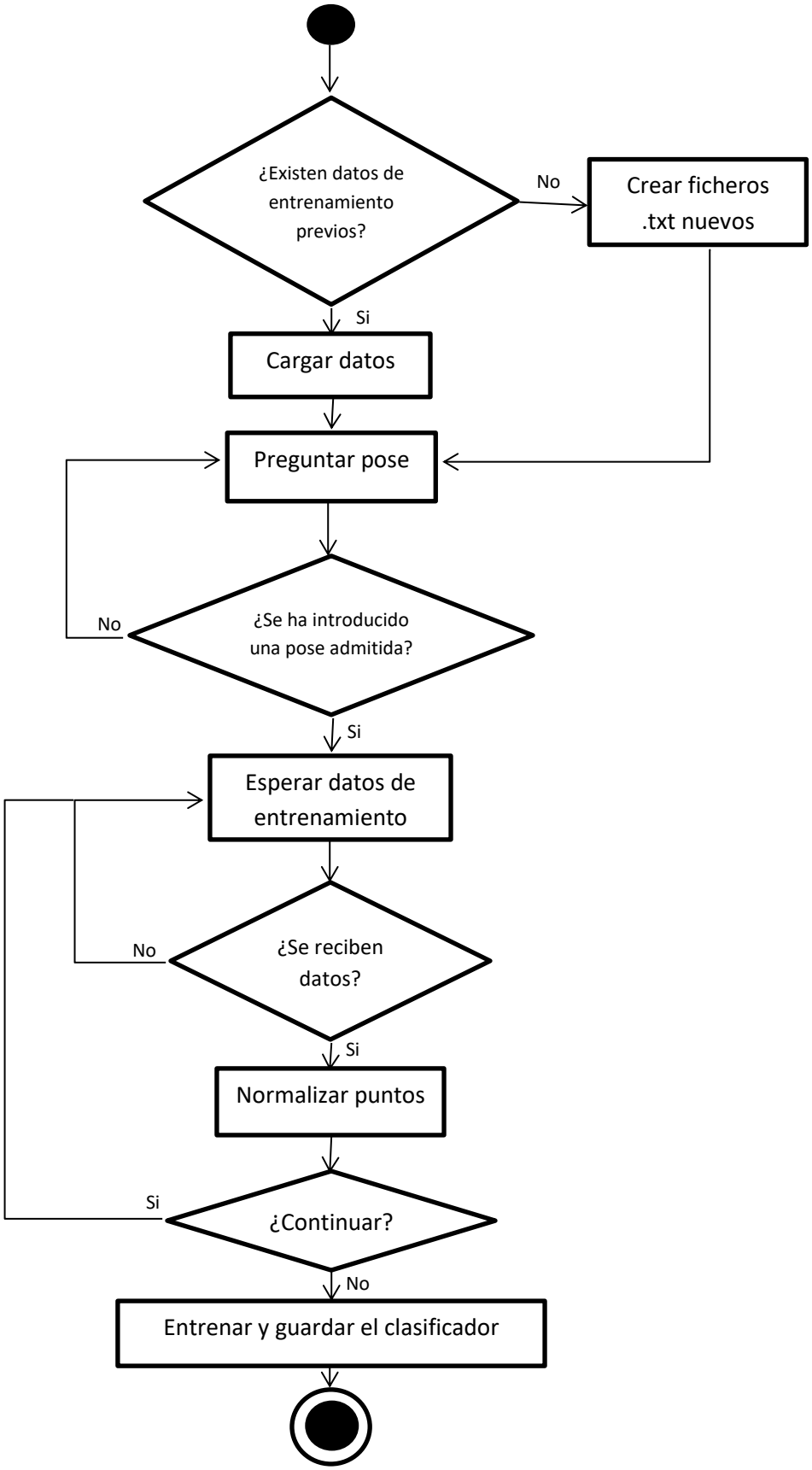


Figura 4.4: Diagrama de flujo del entrenamiento del clasificador

4. **Esperar datos de entrenamiento:** llegado a este punto, el sistema se suscribe a los mensajes que emite el nodo Detección. Estos mensajes se encuentran dentro del tópico "detectionFields", el cual envía puntos en 3D. Una vez suscritos a dicho tópico, el sistema se queda en espera hasta que reciba datos para entrenar el clasificador.

5. **Normalizar puntos:** esta parte se encarga de obtener y manipular los datos de entrenamiento del clasificador. Se obtienen los datos que vienen de los mensajes que envía el nodo Detection, guardándolos en una variable llamada "v_train". Para ello, guardamos uno a uno los 18 puntos con sus respectivas 3 coordenadas, creando un array de 54 valores, que se guardarán sucesivamente en la variable "v_train". Para mejorar la calidad del entrenamiento del clasificador, se normalizan los puntos obtenidos. Primero, guardamos en variables diferentes los valores de las 3 coordenadas (x, y, z) y los ordenamos de mayor a menor. Una vez realizado, utilizando la función "median", obtenemos la mediana de dichos valores, los cuales guardamos en las variables "median_x", "median_y" y "median_z". Tras esto, a todos los valores del array inicial le restamos el valor de la mediana, lo que hace que normalicemos los puntos que caracterizan la boca. Finalmente, una vez normalizados los puntos, los guardamos en el dataset (datasetx) donde cargamos previamente los valores del archivo "file_a.txt". Guardamos en el otro dataset (datasety) el valor numérico de la pose introducida anteriormente. En el caso de que se quiera proseguir con el entrenamiento, el sistema volverá al paso de esperar datos de entrenamiento. Por el contrario, si no se desea seguir con el entrenamiento, se pasará al proceso que describiremos a continuación.

6. **Entrenar y guardar el clasificador:** Por último, llegamos a la parte encargada de entrenar el clasificador que utilizaremos. Para ello, importaremos la biblioteca específica del clasificador que queramos utilizar. Una vez importada la biblioteca, creamos una variable donde guardaremos el objeto relacionado con el clasificador. Utilizando la función fit del objeto clf, podemos entrenar el clasificador utilizando los datasetx y datasety obtenidos en el paso anterior. Una vez entrenado, se procede a guardar los datos del datasetx y del datasety para posteriores entrenamientos, además del archivo correspondiente al propio clasificador para el ejercicio de la terapia.

4.3 Realización de la terapia

Una vez entrenado el sistema, se puede ejecutar el segundo modo de ejecución, la terapia. Esta parte también será programada en lenguaje Python. En la Figura 4.5, podemos ver el diagrama de flujo del proceso de la terapia. A continuación, se explicarán los elementos que conforman dicho diagrama:

1. **Inicio**: al contrario que en el proceso anterior, la adquisición de imágenes se realiza a través de un dispositivo Kinect, en lugar que desde un archivo .bag. Se debe situar al usuario en frente del sensor y arrancar el nodo Detección, el cual detectará el rostro del usuario y tomará los 18 puntos que caracterizan su boca. Estos mensajes se enviarán al tópico "detectionFields" para poder sacar su información en otro nodo distinto.

2. **Preguntar pose**: se pregunta al usuario la pose que desea pronunciar. El sistema comprueba si se ha introducido un valor correcto para continuar con el desarrollo del programa, ya que, en caso de error, se volverá a pedir una pose.

3. **Esperar datos de clasificación:** en este punto, el sistema se suscribe a los mensajes que emite el nodo Detección. Estos mensajes se encuentran dentro del tópico "detectionFields", el cual envía puntos en 3D. Una vez suscritos a dicho tópico, el sistema se queda en espera hasta que reciba datos.

4. **Normalizar puntos:** esta parte se encarga de obtener y manipular los datos para realizar la clasificación. Se obtienen los datos que vienen de los mensajes que envía el nodo Detection, guardándolos en una variable llamada "v_train". Para ello guardamos uno a uno los 18 puntos con sus respectivas 3 coordenadas, creando un array de 54 valores, que se guardarán sucesivamente en la variable "v_train". Para mejorar la calidad del entrenamiento del clasificador, se normalizan los puntos obtenidos. Primero, guardamos en variables diferentes los valores de las 3 coordenadas (x, y, z) y los ordenamos de mayor a menor. Una vez realizado, utilizando la función "median", obtenemos la mediana de dichos valores, los cuales guardamos en las variables "median_x", "median_y" y "median_z". Tras esto a todos los valores del array inicial le restamos el valor de la mediana, lo que hace que normalicemos los puntos que caracterizan la boca. Finalmente, una vez normalizados los puntos, los guardamos en el dataset (datasetx).

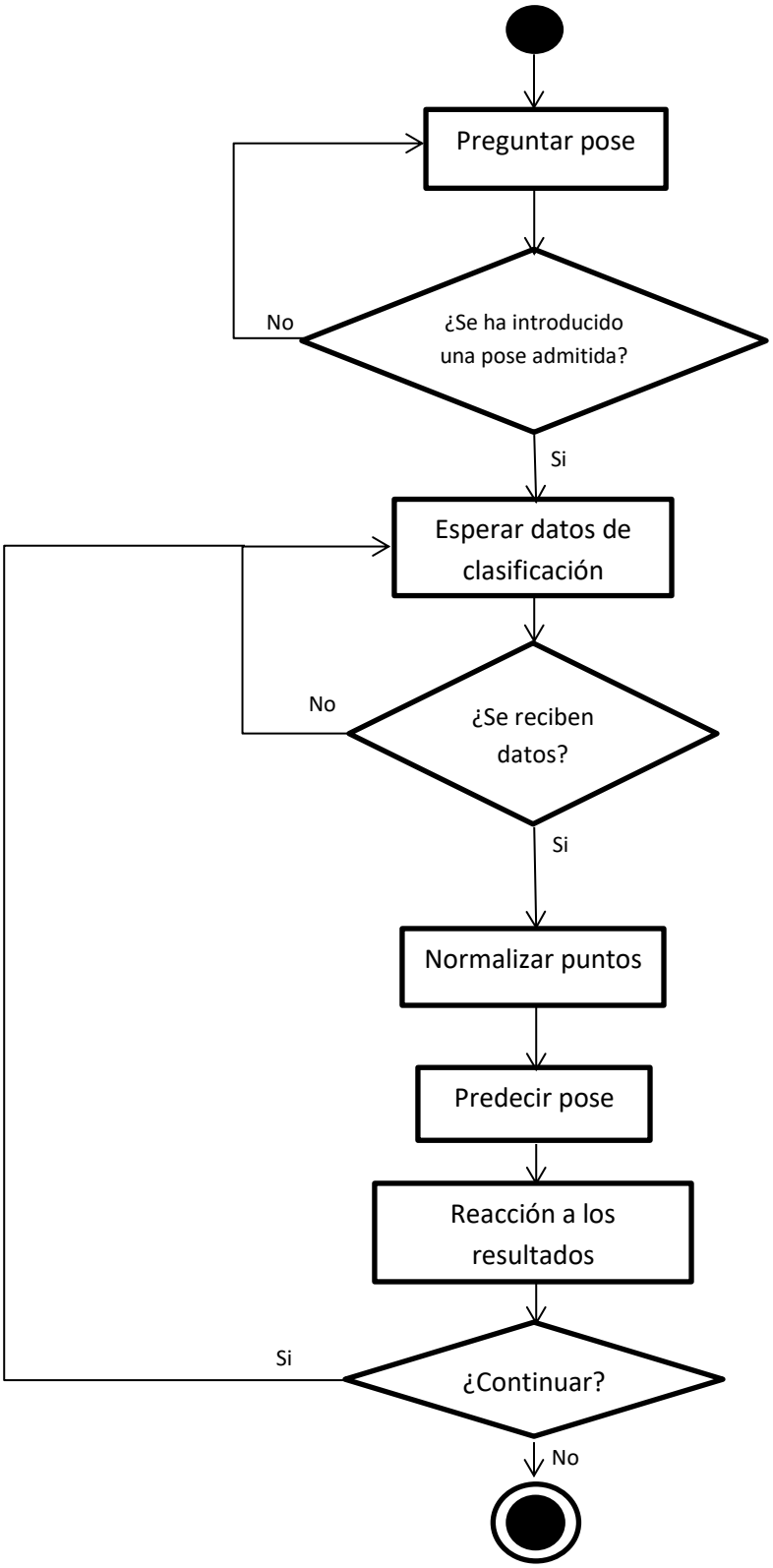


Figura 4.5: Diagrama de flujo de la terapia

5. **Predecir pose:** Esta es la parte donde se realiza la predicción de la pose que está pronunciando el usuario. Mediante la función "predict_proba" que nos ofrece Scikit-Learn, obtenemos la posibilidad en forma de porcentaje de que pose está pronunciando el usuario. La primera pose en llegar a 10 coincidencias se manda al siguiente paso, para proseguir con la lógica.

6. **Reacción a los resultados:** se determina la terapia en sí, utilizando los resultados dados por el paso anterior. El sistema, además de indicar si el usuario está pronunciando mal o bien la pose vocal, da indicaciones de cómo decirla bien. Todo esto lo hace enviando mensajes al tópico "etts" para mandar archivos de audio, al tópico "touch_screen_msgs" para enviar imágenes a la Tablet integrada al robot y al tópico "liveliness_status_update" para enviar cómo se debe mover el robot.

4.4 Interacción con el usuario

Los elementos a estudiar serán: la síntesis de voz del robot, la emisión de imágenes por la Tablet y el propio movimiento del robot.

- **Síntesis de voz (Text-To-Speech(TTS))**: Mediante el nodo "etts", podemos hacer que el robot hable para dar instrucciones al usuario. El principal problema de este elemento es la sincronización, ya que, si el programa va demasiado deprisa, se puede provocar que las frases que debe decir el robot se solapen. En la Tabla 4.1 se explican las frases que emite el robot social.

Tabla 4.1: Frases pronunciadas por el robot

Frase	Situación
Muy bien/ Sigue así/ Lo estás haciendo genial/ Estupendo	Se emplean para felicitar al usuario una vez que ha dicho correctamente la pose pedida
Abre menos la boca/ Abre un poco menos la boca/ Cierra la boca un poquito más	Se emplean para corregir al usuario cuando este tiene la boca más abierta de lo debido durante la vocalización.
Abre más la boca/ Abre un poco más la boca / Abre la boca un poquito más	Se emplean para indicar al usuario que debe abrir más la boca para realizar correctamente el ejercicio
Casi lo tienes/ Estás a punto de conseguirlo	Se emplean para animar al usuario durante el ejercicio
Vamos a decir la vocal a correctamente tres veces/ Vamos a decir la vocal u correctamente tres veces/ Intenta mantener la boca cerrada correctamente 3 veces/ En 3, 2, 1. ¡YA!	Se emplean para introducir al usuario la tarea que debe realizar.
¿Quieres continuar con el juego? / Hasta la próxima	Estas frases se emplean en la fase final del ejercicio, en el momento de determinar si se continua con la terapia.

- **Imágenes de la Tablet:** Mediante el tópico "touch_screen_msgs", se puede enviar al robot qué imagen queremos que muestre por la Tablet. Todo esto funciona mediante una app propia de la Tablet. En la Tabla 4.2 se muestra cuando se utilizan las imagenes que utiliza el robot social (imágenes bajo licencia Creative Commons):

Tabla 4.2: Imágenes mostradas por el robot

Imagen	Situación
	Esta imagen se utiliza para indicar al usuario que debe pronunciar la vocal a.
	Esta imagen se utiliza para decir al usuario que debe tener la boca cerrada.
	Esta imagen se emplea para indicar al usuario que debe pronunciar la vocal u
	Esta imagen se utiliza para felicitar al usuario cuando vocaliza una pose correctamente

- **Movimiento del robot:** Mediante el tópico "liveliness_status_update" enviamos al robot los movimientos que debe realizar. Al robot se le indica los dos estados de ánimo en los que puede estar: triste y contento, y en cada estado de ánimo, el robot se comportará de una manera diferente. En la Tabla 4.3 podemos ver las situaciones en las que se usa cada uno:

Tabla 4.3: Estados de ánimo del robot

Estado	Situación
Contento	El robot se mostrará contento en los casos en los que el usuario realice correctamente el ejercicio de la terapia.
Triste	El robot estará triste cuando el usuario falle a la hora de pronunciar la pose propuesta.

4.5 Implementación de las clases

Para el desarrollo del sistema se han creado 4 clases diferentes: File_processing, Game, train_data y Train. Con estas clases otorgamos diferentes funcionalidades a nuestro sistema, con lo que conseguimos cumplir los objetivos propuestos. Además de estas clases, heredamos la clase Detección del proyecto anterior, que nos da la información de la boca del usuario captada por el sensor Kinect. En la Figura 4.6 podemos ver el diagrama de clases del presente proyecto:

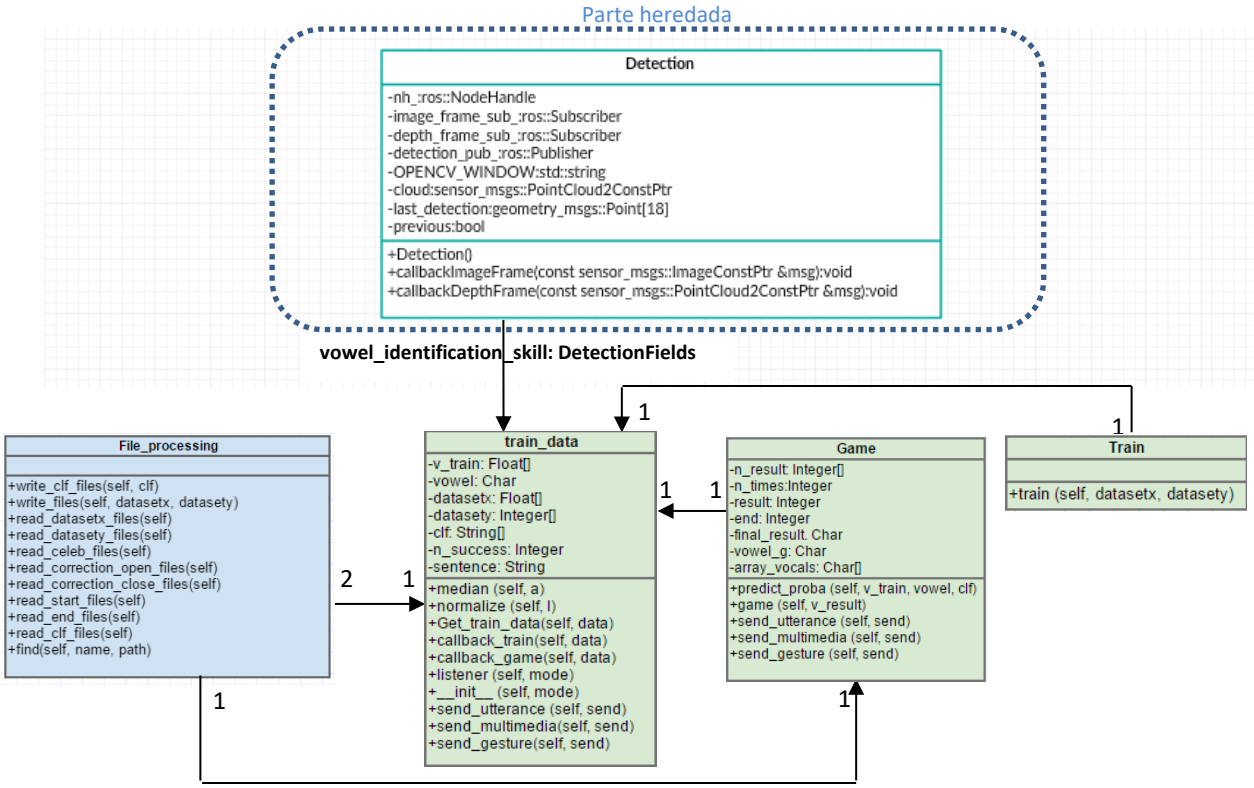


Figura 4.6: Diagrama de clases del sistema

4.5.1 Clase File_processing

En la clase File_processing, se han intentado agrupar todas las funciones que conlleven un manejo de ficheros, ya sea para guardar datos o para cargarlos y llevarlos a otras funciones. Las funciones que se han diseñado son las siguientes:

- *find:* este método es el encargado de encontrar, dentro del sistema, la ruta del archivo que se desea utilizar. Esto es muy útil para la implementación del

proyecto en otros sistemas, ya que nos ahorra tener que cambiar manualmente la ruta de todos los archivos utilizados en el resto de métodos de esta clase.

- *write_clf_file*: este método se encarga de guardar los datos del clasificador entrenado en un fichero llamado "file_clf.txt". Se buscará la ruta del archivo y la utilizaremos para abrir y sobreescribir los datos que posee con los del nuevo entrenamiento. Así, tendremos el clasificador entrenado ya disponible para utilizarlo en otra ocasión, y se previene la pérdida de datos.

- *write_files*: en este método se guardan las variables de entrenamiento del clasificador, el datasetx y el datasety. En el datasetx se guardan todos los puntos utilizados para el entrenamiento, y en el datasety, siguiendo el orden del anterior, se guardan los valores de las poses de los puntos respectivos. Para ello, buscamos la ruta de los archivos "file_a.txt" y "file_b.txt", y utilizándola, abrimos dichos archivos para sobreescribir los datos que posee. Por lo tanto, guardaremos los datos que previamente cargamos junto a los datos que hemos obtenido en este entrenamiento.

- *read_datasetx_files*: Esta función es la encargada de cargar los puntos creados en anteriores entrenamientos del clasificador. En esta función se busca en el sistema algún archivo con el nombre de "file_a.txt". Una vez encontrado, se usa la ruta en la cual está dicho archivo para abrirlo y guardar los datos dentro de una variable, para poder enviar los datos a otras funciones y métodos.

- *read_datasety_files*: Esta función es la encargada de cargar las poses a las que corresponden los puntos guardados en el archivo "file_a.txt". En esta función, al igual que en la anterior, se busca en el sistema algún archivo con el nombre de "file_b.txt". Tras esto, se utiliza la ruta para abrir el archivo y guardar los datos que contiene en una variable para su posterior uso.

- *read_celeb_files*: este método carga las frases utilizadas para felicitar al usuario cuando realiza correctamente el ejercicio. Estas frases pueden ser: "Muy bien" y "Bien hecho". Para realizar la carga de estas frases, se busca la ruta del archivo "celebrations_file.txt", que usaremos para abrir dicho fichero y guardar en una variable el contenido de este. Esta variable se utilizará para mover estos datos de una función a otra.

- *read_correction_open_files*: en este método cargaremos las frases utilizadas para corregir al usuario cuando abra poco la boca en la pronunciación de las poses durante el ejercicio. En la Sección 4.4 podemos ver qué frases se utilizan para señalar al usuario que abra más la boca. Para realizar la carga de las frases,

utilizaremos el método “find” para obtener la ruta del archivo que las guarda. Una vez obtenida la ruta, abrimos el archivo y guardamos los datos que contiene en una variable, para su posterior uso en otra función.

- *read_correction_close_files*: en este método cargaremos las frases utilizadas para corregir al usuario cuando abra demasiado la boca en la pronunciación de las poses durante el ejercicio. En la Sección 4.4 podemos ver qué frases se utilizan para señalar al usuario que abra más la boca. Para realizar la carga de las frases, utilizaremos el método “find” para obtener la ruta del archivo que las guarda. Una vez obtenida la ruta, abrimos el archivo y guardamos los datos que contiene en una variable, para su posterior uso en otra función.

- *read_start_files*: con este método cargamos las frases utilizadas durante el inicio de la terapia (ver Sección 4.4). Se busca en el sistema un archivo con el nombre “start_file.txt” y se guarda en una variable la ruta de dicho archivo. Utilizando esta ruta, abrimos el archivo y guardamos los datos en una variable para poder utilizarlo en otras funciones posteriores.

- *read_end_files*: con este método cargamos las frases utilizadas durante la fase final de la terapia (ver Sección 4.4). Se busca en el sistema un archivo con el nombre “end_file.txt” y se guarda en una variable la ruta de dicho archivo. Utilizando esta ruta, abrimos el archivo y guardamos los datos en una variable para poder utilizarlo en otras funciones posteriores.

- *read_clf_files*: con este método cargamos los datos del clasificador entrenado, que podremos utilizar para realizar la clasificación de los datos que nos lleguen del sensor Kinect. Se busca en el sistema un archivo con el nombre “file_clf.txt” y se guarda en una variable la ruta de dicho archivo. Utilizando esta ruta, abrimos el archivo y guardamos los datos del clasificador en una variable para poder utilizarlo en otras funciones posteriores.

4.5.2 Clase train_data

Esta clase se utiliza como núcleo para toda la lógica. Esta clase contiene el hilo principal donde se llaman al resto de clases y métodos. La función principal de esta clase es la manipulación de los datos que llegan desde el nodo Detectión para después enviarlos al resto de clases. Además, posee los callbacks necesarios para leer los datos del tópico “detectionFields”, donde se guarda la información que llega del nodo Detection. Los métodos que posee esta clase son los siguientes:

- *median*: esta función es la encargada de realizar la mediana de los valores que le entran. Este método devuelve el valor de la mediana resultante.

- *normalize*: este método tiene la función de manipular los datos obtenidos del nodo Detection. Esto nos permite realizar un mejor entrenamiento y obtener unos mejores resultados. Normalizar los puntos de la boca nos permite obviar la posición en la que esté, por lo que nos dará igual que se encuentre en una zona u otra de la pantalla. Primero, guardamos en variables diferentes los valores de las 3 coordenadas (x, y, z) y los ordenamos de mayor a menor. Una vez realizado, utilizando la función "median", obtenemos la mediana de dichos valores, los cuales guardamos en las variables "median_x", "median_y" y "median_z". Tras esto, a todos los valores del array inicial le restamos el valor de la mediana, lo que hace que normalicemos los puntos que caracterizan la boca. Para ello, contamos de tres en tres para restar a las coordenadas x su respectiva mediana, y así con el resto de coordenadas (y, z). Finalmente, una vez normalizados los puntos, los guardamos en el dataset (datasetx) donde cargamos previamente los valores del archivo "file_a.txt", además de guardar en el otro dataset (datasety), donde cargamos previamente los datos del archivo "file_b.txt", los valores de la pose introducida.

- *Get_train_data*: este método se encarga del manejo de los puntos que caracterizan la boca. Primero se guardan los datos de los 18 puntos que vienen de los mensajes que envía el nodo Detection, guardándolos en una variable llamada "v_train". Para ello, guardamos uno a uno los 18 puntos con sus respectivas 3 coordenadas, creando un array de 54 valores, que se guardarán sucesivamente en la variable "v_train" previamente nombrada. Se llama a la rutina "normalize" para normalizar los puntos, lo que nos hace optimizar los resultados del presente proyecto. Acabado esto, guardamos los datos normalizados en una variable y los datos de las poses a las que representan en otra.

- *callback_train*: este es el callback específico del proceso de entrenamiento del clasificador. En este método se llama a la función "Get_train_data" para manipular los puntos que lleguen desde los mensajes que mande el nodo Detection.

- *callback_game*: este es el callback específico del proceso del ejercicio de la terapia. Primero llama al método "Get_train_data" para manipular los datos de los mensajes que llegan del nodo Detection. Una vez obtenidos los datos, llamamos a la función "predict" de la clase Game para seguir con la lógica del proceso de la terapia.

- *listener*: este es el método donde nos suscribimos a los mensajes enviados al tópico “detectionFields”. Estos mensajes son generados por el nodo Detection. Dependiendo del proceso que estemos realizando (entrenamiento o terapia) el método llama a un callback determinado. Si estamos en el proceso de la terapia, llamamos al método “read_clf_files” de la clase File_processing para cargar el clasificador ya entrenado.

- *__init__*: este es el constructor de la clase train_data. Podemos considerarlo como el método base, donde se establece la lógica del sistema. En él, a partir de una variable llamada “mode”, establecemos que lógica seguir dependiendo de si estamos entrenando el clasificador (0) o si estamos realizando la actividad de la terapia (1). Se empieza intentando leer los archivos donde están guardados los puntos creados en entrenamientos anteriores mediante las funciones pertenecientes a la clase File_processing: read_datasetx_files() y read_datasety _files(). Tras utilizar estas funciones y obtener los datos de entrenamientos previos, el sistema pregunta al usuario qué pose vocal se va a entrenar, para poder relacionar correctamente los puntos que se van a guardar con su pose correspondiente. Tras esto, dependiendo del valor de la variable “mode”, se seguirá uno de los dos procesos descritos en las Secciones 4.2 y 4.3.

- *send_utterance*: Este método es el encargado de publicar las frases que debe pronunciar el robot en cada una de las situaciones que ocurran. Similar a la utilizada en la Sección 4.5.3.

- *send_multimedia*: Este método es el que se encarga de publicar qué imágenes deben de verse por la Tablet del robot social Mini. Similar a la utilizada en la Sección 4.5.3.

- *send_multimedia*: Este método es el que se encarga de publicar qué imágenes deben de verse por la Tablet del robot social Mini. Similar a la utilizada en la Sección 4.5.3.

4.5.3 Clase Game

La siguiente clase es la que se encarga de realizar la estimación de qué pose está pronunciando el usuario y de realizar el ejercicio a partir de los resultados obtenidos. Esta clase tiene implementadas las bibliotecas de Scikit-Learn para realizar la

estimación. Posee dos métodos, “predict” y “game”, los cuales explicaremos a continuación:

- *Predict_proba*: Esta es la función donde se realiza la predicción de la pose que está pronunciando el usuario. Mediante la función “predict_proba” que nos ofrece Scikit-Learn, obtenemos la posibilidad en forma de porcentaje de qué pose está pronunciando el usuario. Si la probabilidad de acierto de una pose vocal supera el 35%, el sistema guarda el acierto en el array llamado “n_result”. En el momento en el cual el número de coincidencias de una pose llega a 10, se manda dicha pose al método “game()”. Una vez realizado esto, reiniciamos los valores del array “n_result”.

- *game*: En esta función se determina la lógica de la aplicación, utilizando los resultados finales dados por el paso anterior. El sistema, además de indicar si el usuario está pronunciando mal o bien la pose vocal, da indicaciones de cómo decirlo bien. Todo esto lo hace enviando mensajes al nodo “etts”, donde se manda la frase que debe decir, y al tópico “touch_screen_msgs”, donde se mandan imágenes a la Tablet del robot. En la Figura 4.6 se muestra la lógica que se sigue para realizar este proceso:

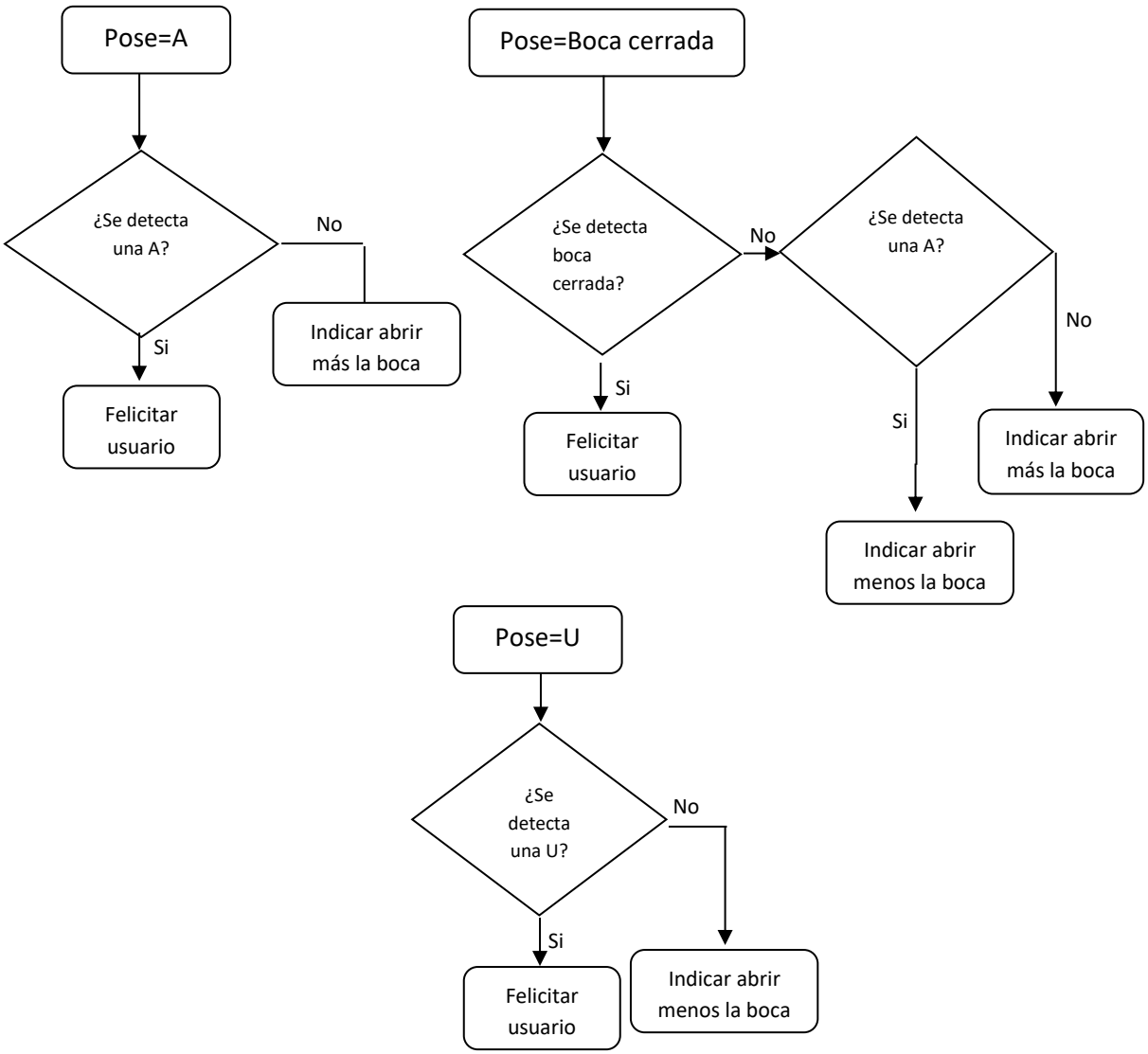


Figura 4.7: Diagramas de flujo de todos los casos de la función Game()

- *send_utterance*: Este método es el encargado de publicar las frases que debe pronunciar el robot en cada una de las situaciones que ocurran. En el caso de acierto, se enviará el mensaje "Bien hecho" para que lo diga el robot. En caso contrario, el robot dirá "Abre más la boca" o "Abre menos la boca" en función de la pose detectada. Dichas frases se publicarán en el tópico "etts", que estarán incluidas en la variable "Utterance".

- *send_multimedia*: Este método es el que se encarga de gestionar qué imágenes deben de verse por la Tablet del robot social Mini. Se enviará la ruta de la imagen a través del mensaje "MultimediaContent" al tópico "multimedia-request". En la Tabla 4.2 podemos ver las imágenes escogidas.

4.5.4 Clase Train

Esta clase es la encargada de realizar el entrenamiento del clasificador. Tiene declaradas las bibliotecas de Scikit-Learn para poder crear y entrenar el clasificador. Posee un único método que describiremos a continuación:

- *train*: Esta es la función encargada de entrenar el clasificador que utilizaremos. Para ello, importaremos la biblioteca específica del clasificador que queramos utilizar. Una vez importada la biblioteca, creamos una variable donde guardaremos el objeto relacionado con el clasificador. Utilizando la función “fit” dada por el clasificador, podemos entrenarlo utilizando los datasetx y datasety, los cuales creamos anteriormente con los datos necesarios.

5. Evaluación del sistema

En este capítulo realizaremos la evaluación del sistema que se ha desarrollado. Se detallará la experimentación realizada, explicando los resultados obtenidos.

5.1 Evaluación del entrenamiento

En esta sección se explicarán los resultados obtenidos a partir de los experimentos realizados para comprobar la calidad del entrenamiento del clasificador. Se detallarán las pruebas realizadas, junto con las conclusiones sacadas de cada una de ellas.

5.1.1 Entrenamiento de las 5 vocales

En esta fase del entrenamiento se entrenó el sistema con muestras de varias personas grabadas en archivos .bag. En cada archivo .bag se guardaban entre 2 y 3 segundos de video donde un usuario pronunciaba una vocal. En total, se utilizaron 59 archivos para el entrenamiento, en los cuales se almacenaban las 5 vocales pronunciadas por diferentes personas. Los archivos de entrenamiento se guardaron para poder realizar las pruebas pertinentes en el futuro.

A la hora de realizar las pruebas, pudimos comprobar que el sistema reaccionaba muy mal a la hora de predecir la vocal que decía el usuario. En la Tabla 5.1 podemos ver los resultados obtenidos con cada uno de los clasificadores utilizados:

Tabla 5.1: Resultados del entrenamiento de las 5 vocales

CLASIFICADOR	RESULTADOS
K-NEAREST NEIGHBORS	Solo se detectaba correctamente la a
RANDOM FOREST	Detectaba ligeramente las vocales a(15%)-i(13%)
SVM	Únicamente daba como resultado la vocal e
DECISION TREE C4.5	Detectaba ligeramente las vocales e(22%)-u(12%)

A partir de estos resultados, se decidió suprimir el clasificador SVM de las pruebas futuras, debido a los pésimos resultados que daba. Además, se planteó la idea de reducir el número de vocales a entrenar para observar cómo responde el clasificador.

5.1.2 Entrenamiento de 2 vocales

Llegado a este punto, se decidió entrenar el clasificador con 2 vocales y las vocales escogidas fueron la a y la u. Se llegó a esta decisión debido a la similitud que existe en la pronunciación de algunas vocales. Por ejemplo, la pronunciación de la vocal a y la vocal e es muy similar, al igual que la pronunciación de la vocal o y la vocal u en algunos casos. En la Figura 5.1 y 5.2 podemos ver las similitudes de dichas vocales. Debido a esto, se entrenaron las vocales a-u debido a que son fácilmente diferenciables, ya que una se pronuncia con la boca abierta y la otra se pronuncia con la boca bastante más cerrada.

Figura 5.1: Vocalización de las vocales a(derecha)-e(izquierda)

Figura 5.2: Vocalización de las vocales o(derecha)-u(izquierda)

Para el entrenamiento de estas dos vocales, se emplearon los archivos .bag utilizados anteriormente, pero en este caso sólo aquellos correspondientes a las vocales a-u. En total eran 22 archivos con una duración media de 2 segundos de video. En este caso los resultados también fueron muy poco prometedores, ya que el sistema no era capaz de predecir correctamente la vocal pronunciada. Esta parte del entrenamiento

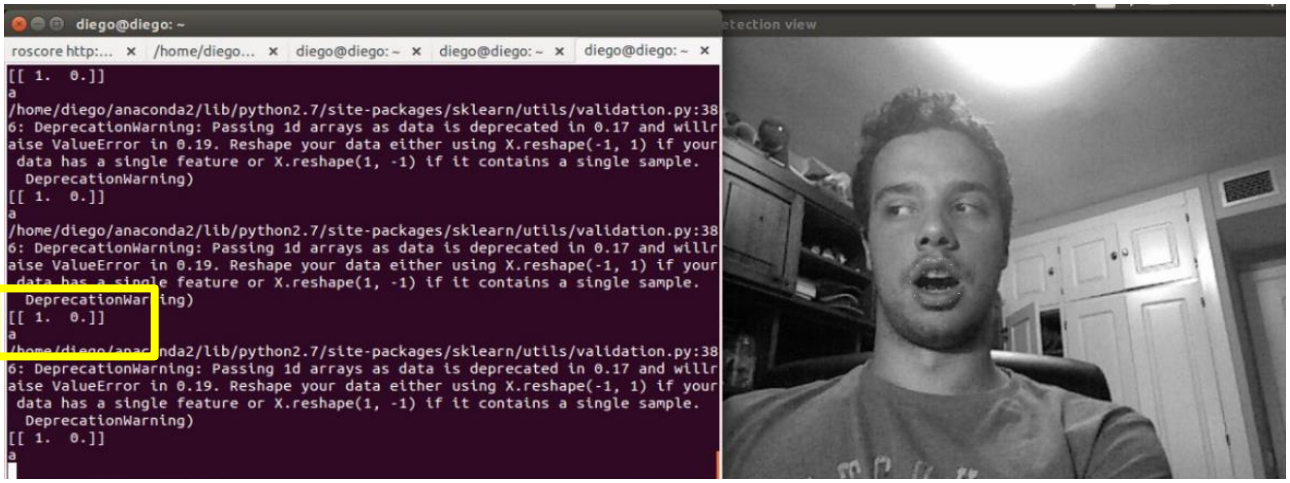
podemos calificarla como entrenamiento off-line, donde utilizamos archivos pregrabados para el entrenamiento del sistema.

A partir de este punto se intentó buscar una solución a los malos resultados que se obtenían de los clasificadores. Se llegó a la conclusión de que el conjunto de datos utilizados para entrenar era muy pequeño. El sistema necesita una gran cantidad de muestras para poder estimar con un porcentaje de aciertos aceptable. Este es el momento en el que entramos en el entrenamiento on-line, donde utilizamos videos en directo para el entrenamiento del clasificador. Este entrenamiento es mucho más delicado que el off-line, ya que se requiere que la boca del usuario esté siendo correctamente detectada para no entrenar el sistema con datos erróneos.

Para la ejecución de dicho entrenamiento, se ejecuta el nodo que realiza la clasificación para comprobar si se llega a detectar correctamente la vocal pronunciada. En caso negativo, se ejecuta el nodo de entrenamiento para entrenar dicha vocal. Se realiza este procedimiento tantas veces como sean necesarias hasta conseguir los resultados deseados. Para entrenar estas dos vocales, se ha dedicado alrededor de 10 horas para conseguir unos resultados aceptables (ver Tabla 5.2).

Se pudo observar que importa en gran medida la distancia y el ángulo de la boca con respecto a la cámara para una correcta clasificación. Debido a esto, el proceso de entrenamiento se hizo mucho más largo y tedioso que el realizado con el entrenamiento off-line, ya que se tenía que intentar abarcar la mayoría de las posiciones que el usuario pudiese tomar.

Conforme se iba entrenando y probando el clasificador, se observó que el que mejor respondía a los entrenamientos era el Decision Tree C4.5. En la Figura 5.3 y 5.4 podemos observar como respondía el sistema a las vocales pronunciadas por el usuario:



```
[[ 1.  0.]]
a
/home/diego/anaconda2/lib/python2.7/site-packages/sklearn/utils/validation.py:38
6: DeprecationWarning: Passing 1d arrays as data is deprecated in 0.17 and willr
aise ValueError in 0.19. Reshape your data either using X.reshape(-1, 1) if your
 data has a single feature or X.reshape(1, -1) if it contains a single sample.
  DeprecationWarning)
[[ 1.  0.]]
a
/home/diego/anaconda2/lib/python2.7/site-packages/sklearn/utils/validation.py:38
6: DeprecationWarning: Passing 1d arrays as data is deprecated in 0.17 and willr
aise ValueError in 0.19. Reshape your data either using X.reshape(-1, 1) if your
 data has a single feature or X.reshape(1, -1) if it contains a single sample.
  DeprecationWarning)
[[ 1.  0.]]
a
/home/diego/anaconda2/lib/python2.7/site-packages/sklearn/utils/validation.py:38
6: DeprecationWarning: Passing 1d arrays as data is deprecated in 0.17 and willr
aise ValueError in 0.19. Reshape your data either using X.reshape(-1, 1) if your
 data has a single feature or X.reshape(1, -1) if it contains a single sample.
  DeprecationWarning)
[[ 1.  0.]]
a
```

Figura 5.3: Respuesta del sistema al vocalizar una a

```
u
/home/diego/anaconda2/lib/python2.7/site-packages/sklearn/utils/validation.py:38
6: DeprecationWarning: Passing 1d arrays as data is deprecated in 0.17 and willr
aise ValueError in 0.19. Reshape your data either using X.reshape(-1, 1) if your
 data has a single feature or X.reshape(1, -1) if it contains a single sample.
  DeprecationWarning)
[[ 0.  1.]]
u
/home/diego/anaconda2/lib/python2.7/site-packages/sklearn/utils/validation.py:38
6: DeprecationWarning: Passing 1d arrays as data is deprecated in 0.17 and willr
aise ValueError in 0.19. Reshape your data either using X.reshape(-1, 1) if your
 data has a single feature or X.reshape(1, -1) if it contains a single sample.
  DeprecationWarning)
[[ 0.  1.]]
u
/home/diego/anaconda2/lib/python2.7/site-packages/sklearn/utils/validation.py:38
6: DeprecationWarning: Passing 1d arrays as data is deprecated in 0.17 and willr
aise ValueError in 0.19. Reshape your data either using X.reshape(-1, 1) if your
 data has a single feature or X.reshape(1, -1) if it contains a single sample.
  DeprecationWarning)
[[ 0.  1.]]
u
```

Figura 5.4: Respuesta del sistema al vocalizar una u

En la Tabla 5.2 podemos ver los resultados obtenidos tras el entrenamiento:

Tabla 5.2: Matriz de confusión de los resultados del entrenamiento de 2 vocales

Vocal deseada **Vocal detectada**	**A**	**U**
A	78%	32%
U	22%	68%

Una vez conseguida una detección correcta de las vocales a-u, se decide ampliar las posibilidades de clasificación del sistema, incluyendo más vocales o poses.

5.1.3 Entrenamiento de 3 poses vocales

Llegados a este punto, para mejorar las prestaciones del clasificador, se buscó una pose vocal que no diese problemas de similitud con las dos vocales previamente entrenadas. Dado que el software utilizado para la detección facial funciona muy bien con rostros serios, se decidió que la tercera pose fuera con la boca cerrada.

Para el entrenamiento del sistema, se realizó el llamado entrenamiento on-line, ya que es mucho más rápido y no requiere guardar los archivos .bag utilizados. Se tuvo que dedicar un mayor tiempo de entrenamiento para conseguir que el sistema reconociese dicha vocal, llegado a dedicar aproximadamente unas 15 horas.

En la Figura 5.5, 5.6 y 5.7 podemos ver los resultados finales del entrenamiento:

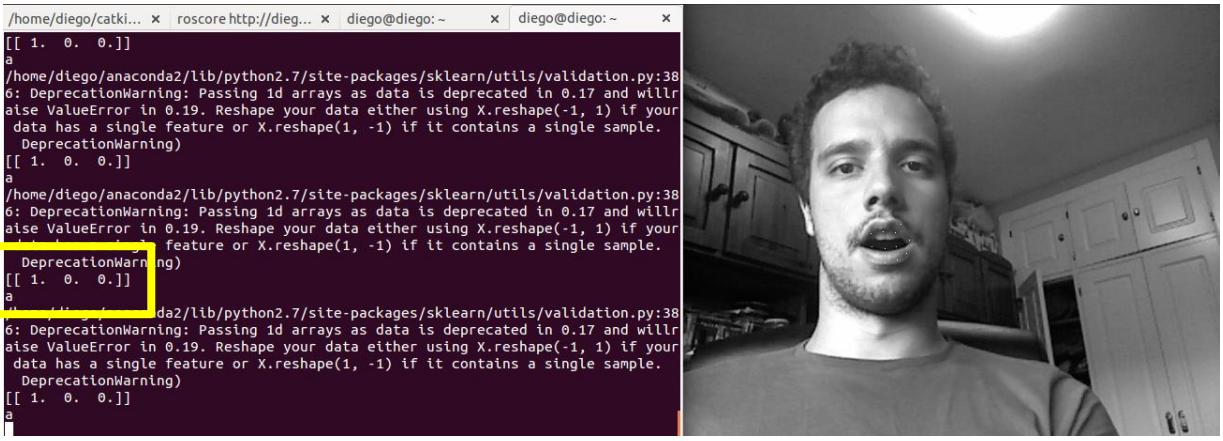


Figura 5.5: Respuesta del sistema al vocalizar una a

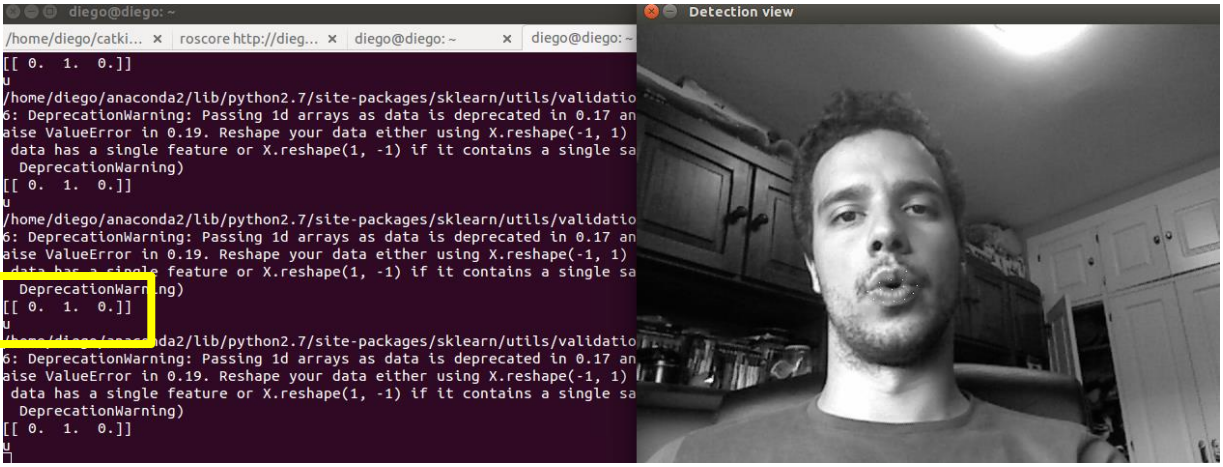


Figura 5.6: Respuesta del sistema al vocalizar una u

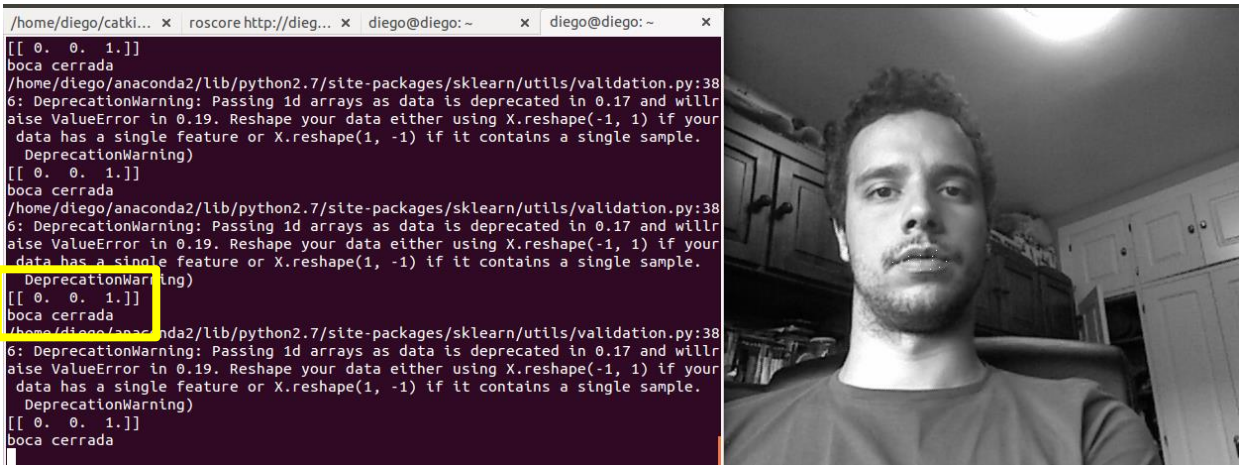


Figura 5.7: Respuesta del sistema al detectar la boca cerrada

Además, en la Tabla 5.3 podemos observar la matriz de confusión de los resultados de dicho experimento:

Tabla 5.3: Matriz de confusión de los resultados del entrenamiento de 3 poses vocales

Pose deseada / Pose detectada	A	U	Boca Cerrada
A	81%	10%	1%
U	15%	74%	17%
Boca Cerrada	4%	16%	82%

Una vez finalizado este paso, podemos pasar a la evaluación de la interfaz y los experimentos con personas ajenas al entrenamiento.

5.2 Evaluación de los experimentos

Comprobado el funcionamiento del sistema con rostros empleados en el entrenamiento previo, podemos realizar las pruebas con usuarios sin entrenar para observar el comportamiento del sistema. Todos los experimentos previos fueron realizados con rostros utilizados para el entrenamiento del clasificador. En la Figura 5.8 podemos ver un esquema de cómo se han realizado los experimentos.

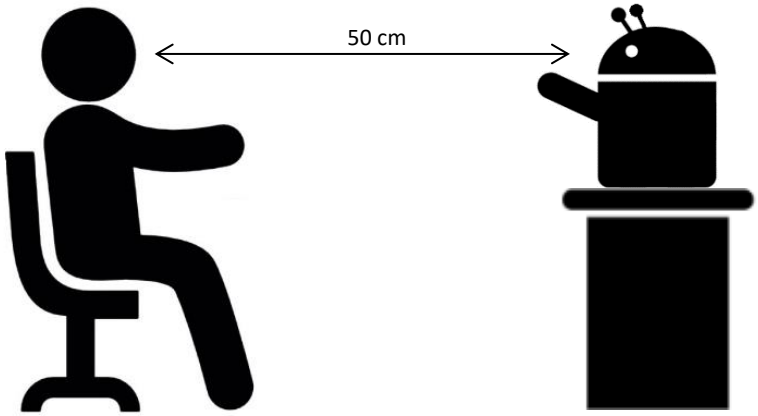


Figura 5.8: Esquema de los experimentos realizados

Los experimentos podemos dividirlos en: experimentos sin entrenamiento y experimentos con entrenamiento. Primero se realizan los experimentos con usuarios con lo que no se ha entrenado el clasificador, para observar cómo responde el sistema. Estos experimentos varían mucho, ya que, dependiendo del tipo de boca y su parecido con las que fueron usadas para entrenar el sistema, los resultados salen muy favorables o poco favorables. Un ejemplo son las pruebas con mujeres, donde no obtenemos ningún resultado positivo, debido a que el entrenamiento inicial fue realizado con

hombres. En la Tabla 5.4 podemos ver la matriz de confusión de los experimentos sin entrenamiento:

Tabla 5.4: Matriz de confusión de los resultados del experimento sin entrenamiento

Pose detectada \ Pose deseada	A	U	Boca Cerrada
A	52%	36%	0%
U	32%	40%	90%
Boca Cerrada	16%	24%	10%

Como podemos observar, los resultados obtenidos no son aceptables, ya que no hay ninguna pose que supere el 60 % de aciertos. Esto es debido a que el sistema necesita entrenarse con una gran cantidad de gente para poder tener unos resultados que se puedan considerar buenos (por encima del 70%). Una vez realizado este experimento, procedemos al entrenamiento de los usuarios para comprobar cómo responde el sistema. Al haber entrenado el sistema previamente durante aproximadamente 15 horas (ver Sección 5.1.3), los entrenamientos son muy breves (aproximadamente 10 minutos). El sistema ya tiene unas referencias sobre las poses que se van a realizar y, por tanto, no es necesario tanto tiempo para obtener unos resultados aceptables. También se obtienen resultados muy favorables en los experimentos con mujeres tras entrenar el clasificador con sus rostros. Se requiere una mayor cantidad de tiempo, ya que el clasificador fue entrenado con rostros de hombres. En la Tabla 5.5 podemos ver los resultados de las pruebas de los experimentos con entrenamiento:

Tabla 5.5: Matriz de confusión de los resultados del experimento con entrenamiento

Pose detectada \ Pose deseada	A	U	Boca Cerrada
A	85%	8%	0%
U	12%	75%	14%
Boca Cerrada	3%	17%	86%

Como se puede ver, la mejora de aciertos es significativa. Cabe destacar que todos los experimentos fueron realizados a 50 centímetros del sensor Kinect, y toda prueba que se realice a otra distancia tendrá unos resultados completamente distintos. Los resultados más favorables se consiguen a esta distancia, y si nos alejamos o acercamos del sensor, los resultados empeoran considerablemente.

En las Figuras 5.9, 5.10, 5.11 y 5.12 se muestra una evaluación cualitativa de los resultados obtenidos tras el entrenamiento de un usuario, así como el mecanismo de nuestra propuesta de terapia:

En la Figura 5.9 podemos ver el inicio del sistema. Se pregunta por terminal qué pose vocal queremos que pronuncie el usuario. Una vez introducida, pasamos a la siguiente fase.

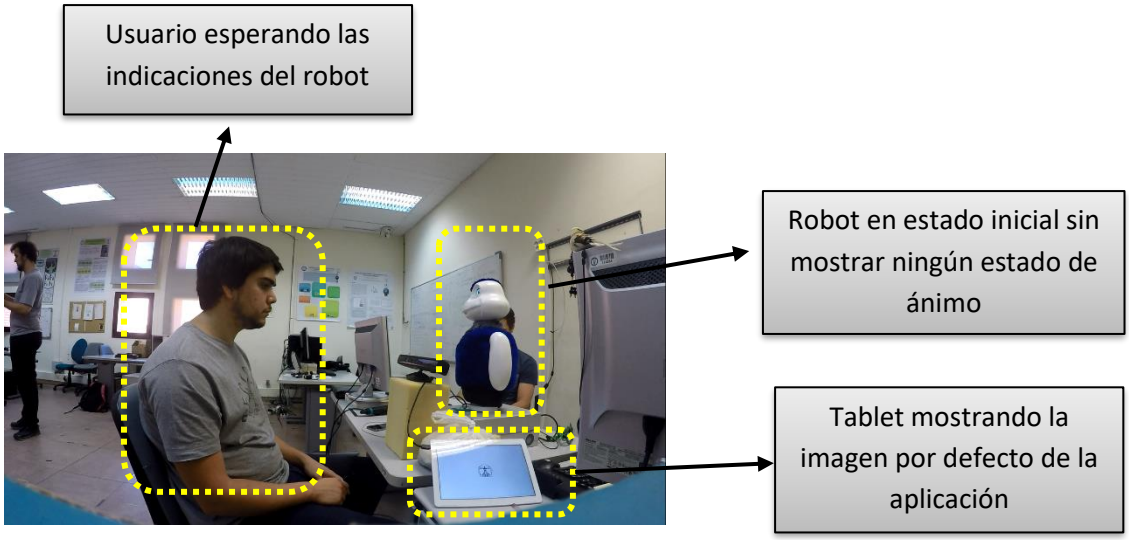


Figura 5.9: Estado inicial del ejercicio

En la Figura 5.10 se puede ver el inicio del ejercicio una vez introducida la pose vocal. El robot indica por voz y por imágenes en la Tablet cual es la pose elegida. Una vez realizado esto, se da la señal al usuario para comenzar la actividad.

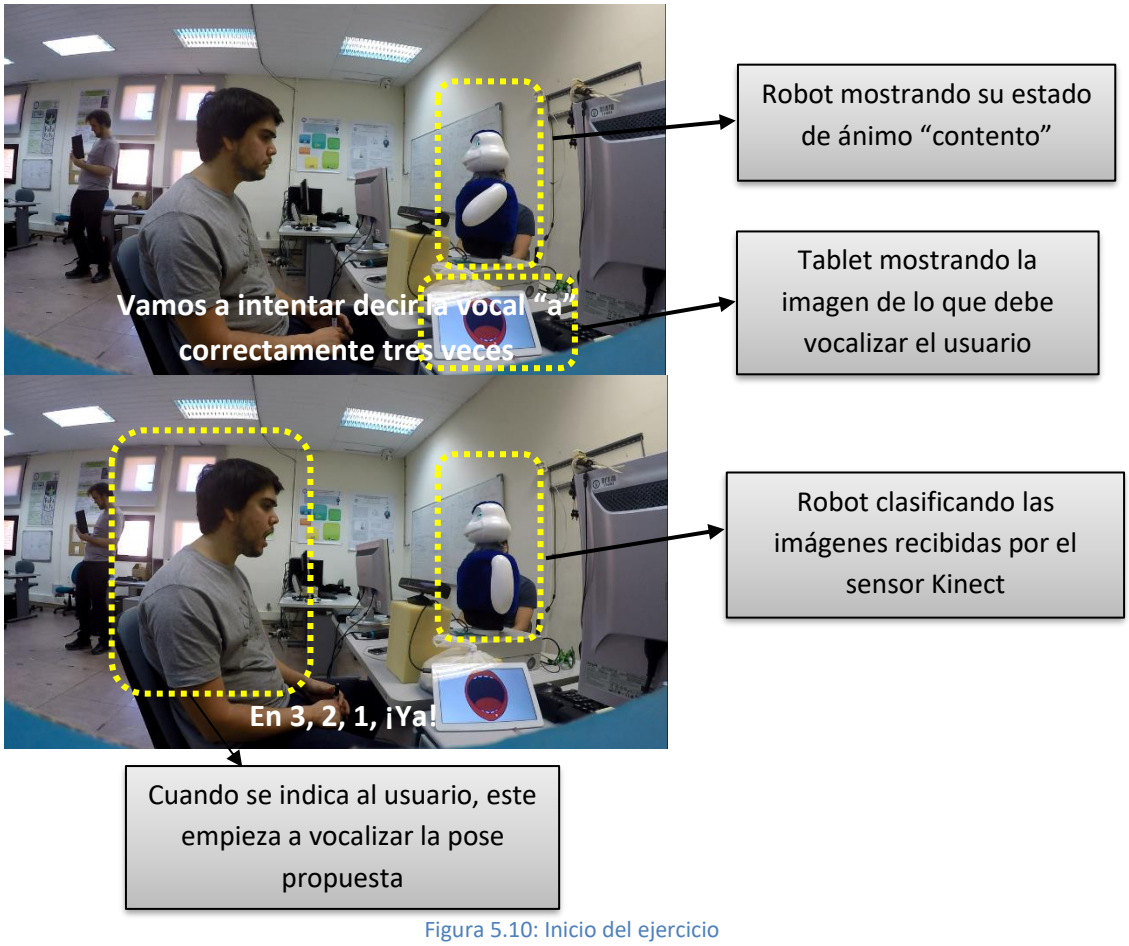


Figura 5.10: Inicio del ejercicio

En la figura 5.11 podemos ver las reacciones del robot social Mini. El robot podrá felicitar al usuario por una correcta vocalización, o corregirle en el caso de que falle con la pose vocal. Se darán las instrucciones pertinentes por medio de la síntesis de voz y mediante imágenes mostradas por la Tablet.

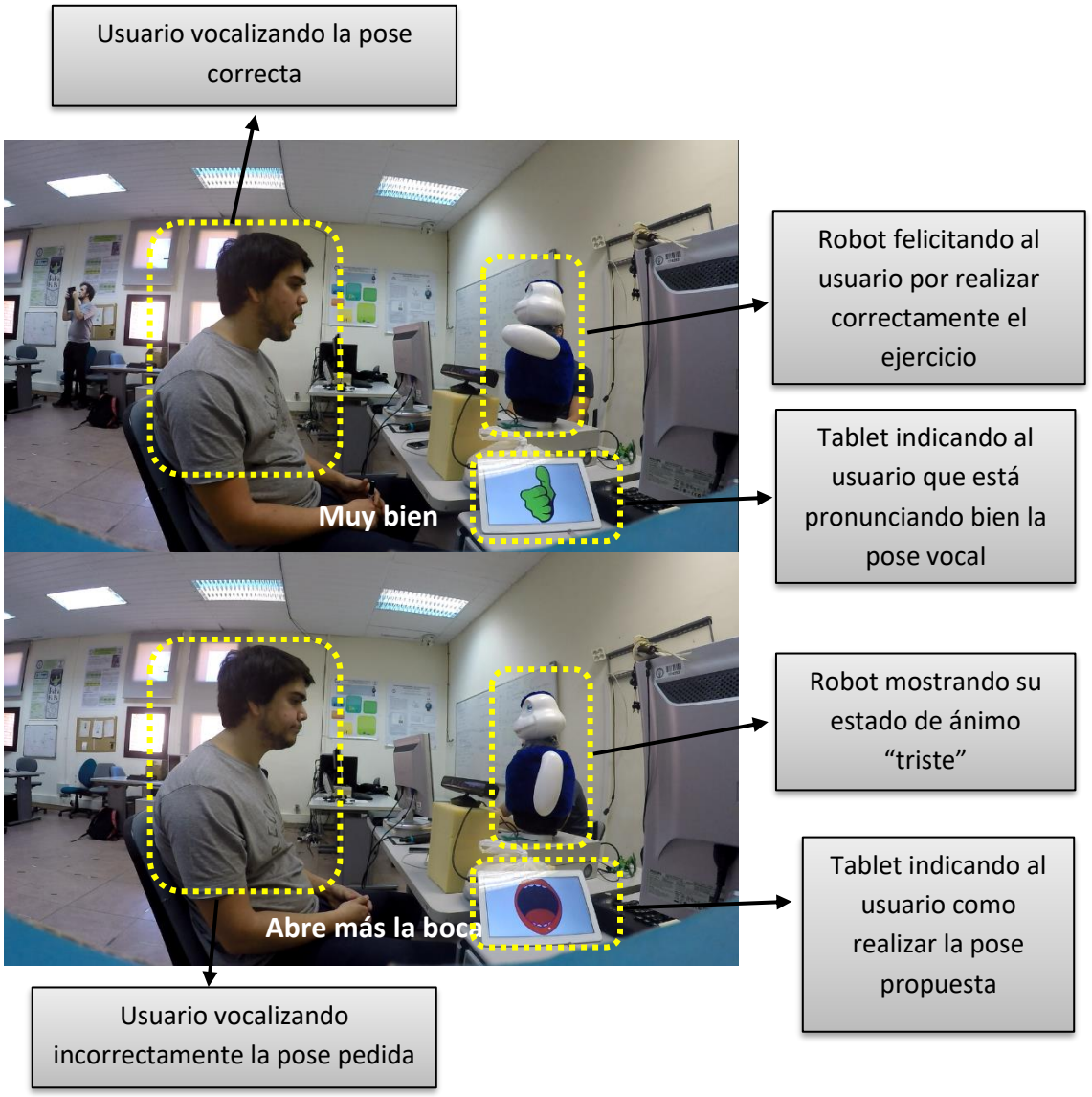


Figura 5.11: Reacciones del robot

Por último, en la Figura 5.11 podemos ver la fase final del ejercicio. Una vez que el robot detecte 3 veces la vocal correcta, nos preguntará si deseamos continuar con el juego, lo cual debemos responder por la terminal. Si deseamos continuar, el robot vuelve a la fase mostrada en la Figura 5.10 y, en caso contrario, se despide de nosotros y finaliza la actividad.

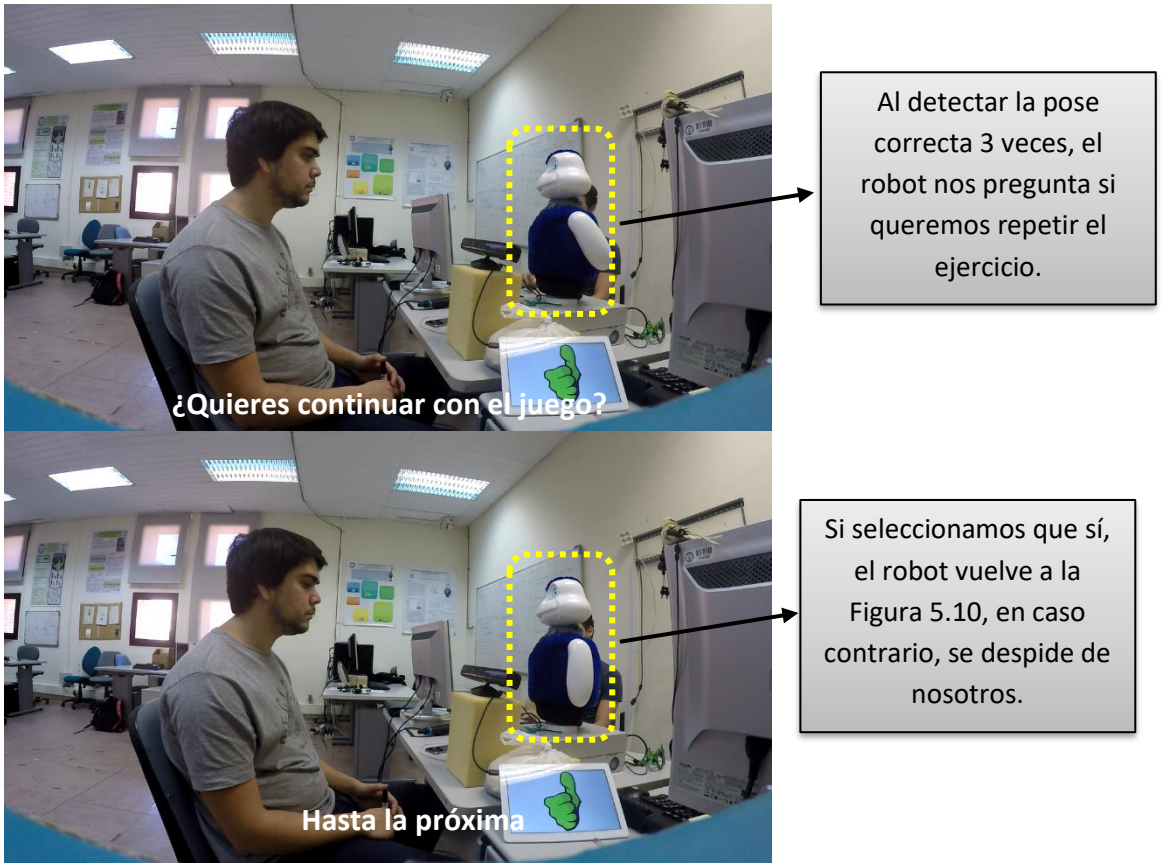


Figura 5.11: Fin del ejercicio

6. Planificación y presupuesto

El siguiente capítulo se dividirá en tres partes: la primera, donde determinaremos la metodología utilizada para la planificación del proyecto, la segunda, donde detallaremos la planificación y la tercera, donde analizaremos el coste asociado.

6.1 Metodología de la planificación

El presente proyecto sigue un modelo cascada para su planificación. En este modelo se ordenan rigurosamente las etapas del proceso para el desarrollo de software, de tal manera que el inicio de cada etapa debe esperar a la finalización del proceso anterior. Este proyecto está organizado de tal manera que al final de cada etapa se realiza una revisión final, por la que se establece si el sistema está preparado para avanzar a la siguiente fase. Se ha elegido este modelo debido a que es idóneo para procesos con poco capital y escasas herramientas. Además, es fácil de aprender e implementar y su uso está muy extendido dentro de la ingeniería del software. La planificación mediante el modelo cascada consta de las siguientes fases:

- **Análisis de requisitos**: en esta etapa se estudian las necesidades de los usuarios finales, determinando qué objetivos se deben alcanzar. En este caso, el tutor del proyecto realizará las funciones de usuario, dándonos las nociones necesarias para establecer los objetivos del sistema. De esta etapa se obtiene un documento de especificación de requisitos, que contiene los objetivos del sistema sin llegar a profundizar en detalles internos.

- **Diseño del sistema**: en esta fase, se descompone y organiza el sistema en los elementos que puedan elaborarse por separado. También, se obtienen las especificaciones de cada parte del sistema y la manera de combinar cada una de ellas. En esta etapa llegamos a definir la arquitectura del sistema.

- **Implementación del sistema**: basándonos en el diseño del sistema, escribimos los algoritmos necesarios para realización del proyecto y realizamos las pruebas necesarias para corregir errores de programación.

- **Pruebas**: en esta etapa, se reúnen todos los elementos del sistema para comprobar si el sistema completo funciona correctamente. Se debe comprobar que se cumplen los objetivos antes de entregar el resultado final.

- **Verificación y mantenimiento**: como etapa final, el usuario prueba el resultado final. Se debe tener el sistema activo para modificar el proyecto en el caso de que surja insatisfacción por parte del usuario.

En este proyecto debe de incluirse una fase de documentación, donde elaboramos la memoria del proyecto. Esta fase la iniciamos durante la fase de análisis del sistema, donde comenzamos a anotar todo lo referente al desarrollo del proyecto. Esta fase no está indicada en la metodología original del modelo, sin embargo, es completamente necesaria. Además, se debe destacar que en la fase de implementación se utilizó una metodología basada en prototipos, como ya se mencionó en la Sección 3.5. En la Figura 6.1 podemos ver un diagrama de la metodología en cascada:

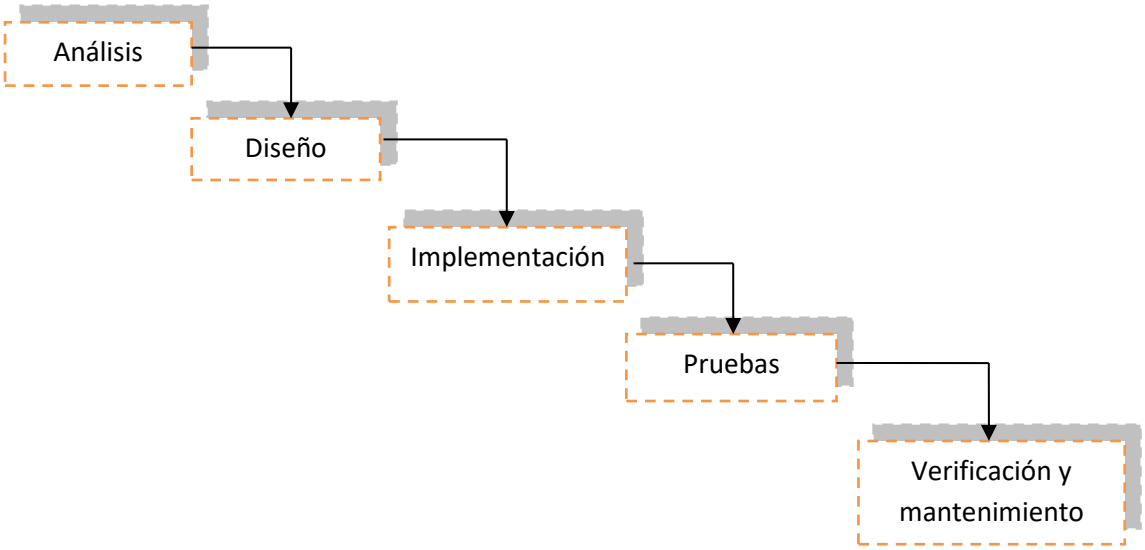


Figura 6.1: Metodología en cascada

6.2 Planificación

En esta sección se detallará la planificación realizada para el presente proyecto, añadiendo sus fases y tareas. Además, estimaremos las horas de dedicación a la semana.

Este proyecto tiene una duración aproximada de 6 meses, comenzando el 15 de marzo de 2016 y finalizando el 25 de septiembre de 2016. Se indica en la Tabla 6.1 las fases definidas en la sección anterior, enumerando cada una de las tareas que definen cada fase, junto a la duración de cada una.

Tabla 6.1: Planificación del proyecto

TAREAS	DURACIÓN (DÍAS)	FECHA INICIO	FECHA FIN
Análisis del sistema	**51**	**15/03/2016**	**4/05/2016**
-Definición de los objetivos del proyecto	4	15/03/2016	18/03/2016
-Documentación ROS	17	19/03/2016	4/04/2016
-Documentación proyecto anterior	10	5/04/2016	14/04/2016
-Documentación Scikit-Learn	5	15/04/2016	19/04/2016
-Documentación Machine-Learning	8	20/04/2016	27/04/2016
-Definición de requisitos	4	28/04/2016	1/05/2016
-Definición de casos de uso	3	2/05/2016	4/05/2016
Diseño del sistema	**21**	**5/05/2016**	**25/05/2016**
-Definición de la arquitectura del sistema	12	5/05/2016	16/05/2016
-Definición del entrenamiento del clasificador	4	17/05/2016	20/05/2016
-Definición de la terapia	5	21/05/2016	25/05/2016
Implementación del sistema	**67**	**26/05/2016**	**31/07/2016**
-Integración ROS en el sistema	6	26/05/2016	30/05/2016
-Integración Stasm	3	31/05/2016	2/06/2016
-Comprobación funcionalidad proyecto anterior	4	3/06/2016	6/06/2016
-Integración Scikit-Learn y Python	4	7/06/2016	10/06/2016
-Desarrollo nodo de entrenamiento	20	11/06/2016	30/06/2016
-Comunicación nodo entrenamiento con nodo Detección	6	1/07/2016	6/07/2016
-Desarrollo nodo de terapia	19	7/07/2016	25/07/2016
-Desarrollo interfaz	6	26/07/2016	31/07/2016
Pruebas	**15**	**1/09/2016**	**15/09/2016**
-Ejecución de pruebas	15	1/09/2016	15/09/2016
Verificación del sistema	**8**	**16/09/2016**	**23/09/2016**
Documentación	**162**	**15/03/2016**	**23/09/2016**

Una vez realizado el estudio de las fases a realizar y los días necesarios para cada una, podemos realizar un diagrama de Gantt. Con este diagrama podremos tener consciencia del uso del método cascada para la planificación del proyecto. Podemos observar que, hasta no finalizar una tarea, no pasaremos a la siguiente, salvo en el caso de la documentación, que se realiza constantemente. En la Figura 6.1 podemos ver el diagrama de Gantt asociado al proyecto:

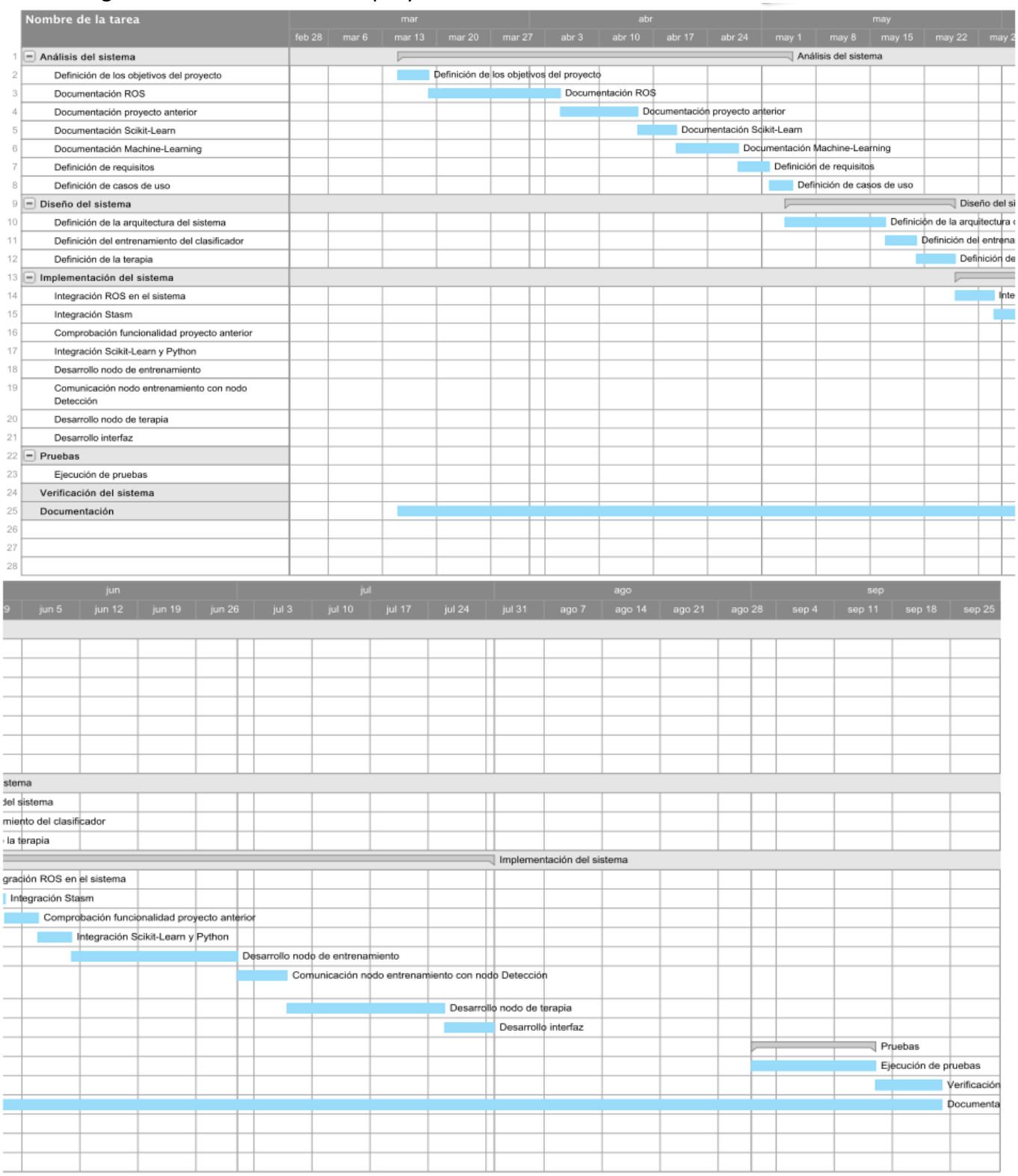


Figura 6.2: Diagrama Gantt

Podemos observar que, en el mes de agosto, al estar la universidad cerrada, y por ello el acceso al laboratorio, se ha dado mayor importancia a la redacción de la documentación que al desarrollo del proyecto. Ya establecida la planificación del proyecto, en la Tabla 6.2 podemos ver una estimación de tiempo dedicada al proyecto durante cada uno de los meses de trabajo. En esta tabla podemos ver como se ha repartido el trabajo a lo largo del periodo de trabajo, y que el total de horas empleadas han sido aproximadamente 520 horas:

Tabla 6.2: Reparto de horas semanales

Mes	Horas a la semana	Total de horas
Marzo	15	60
Abril	15	60
Mayo	10	40
Junio	10	40
Julio	30	120
Agosto	30	120
Septiembre	20	80
Total		**520**

6.3 Presupuesto

En la siguiente sección se determinará el coste asociado a la realización del proyecto. Para la elaboración del presupuesto, se han tenido en cuenta los diferentes tipos de coste, así como la duración del proyecto expuesta en el apartado anterior. Los costes que se han tenido en cuenta son los siguientes:

- Costes de personal
- Costes de material
- Costes indirectos

La unidad monetaria utilizada ha sido el euro (€), realizando un redondeo de dos decimales cuando corresponda. El total del presupuesto debe ser coherente para llegar a ser atractivo para un posible cliente, y debe de realizarse una correcta estimación para evitarle costes adicionales.

6.3.1 Coste de personal

El coste del personal se ha calculado utilizando las tablas salariales facilitadas por el convenio colectivo de ámbito estatal para los centros de educación universitaria e investigación del BOE número 36 del 30 de enero de 2015 [28]. Para el cálculo del coste, se ha tenido en cuenta el rol y las horas de trabajo del personal dedicado a la realización del proyecto. Los roles que tomarán las personas involucradas en la elaboración del sistema serán:

- **Jefe de proyecto**: será la persona encargada de la dirección del proyecto. Tendrá la función de dirigir y coordinar al resto del personal.
- **Analista/Diseñador**: será el encargado de definir los requisitos del sistema y diseñar la arquitectura del proyecto.
- **Programador**: será el encargado de escribir el código e implementar la arquitectura diseñada por el analista.
- **Técnico de pruebas**: realizará las pruebas pertinentes al sistema y será el encargado de determinar la validez del proyecto.

Determinados los roles, en la Tabla 6.3 podemos ver el presupuesto del coste de personal asociado al presente proyecto:

Rol	Meses trabajados (1/2 jornada)	Salario al mes (€)	Total proyecto (€)
Analista/Diseñador	2	1.548,37 €	1.548,37 €
Programador	4	1.202,80 €	2.405,60€
Técnico de pruebas	1	1.074,05 €	537,03 €
Jefe de proyecto	6	1.586,54 €	4.759,62€
TOTAL			**9.250,62€**

Tabla 6.3: Costes de personal

6.3.2 Costes de materiales

En esta sección se estudiarán los costes asociados a los materiales necesarios para el desarrollo del proyecto. Se tendrán en cuenta el precio del producto, periodo de amortización, su uso y el coste total. El coste total de cada producto será calculado mediante la siguiente ecuación:

$$coste = \frac{precio}{periodo\ amortización}$$

En la Tabla 6.4 se puede observar el estudio realizado del coste asociado a los materiales del presente proyecto:

Tabla 6.4: Costes de materiales

Producto	Precio (€)	Periodo Amortización (Meses)	Uso (Meses)	Coste Proyecto (€)
Kinect XBOX 360	60 €	24	6	15 €
PC Samsung i3	300 €	36	6	50 €
Framework ROS	0 €	24	6	0 €
Licencia Microsoft Office	279 €	24	6	69,75 €
Licencia Scikit-Learn	0 €	24	6	0 €
Licencia Stasm	0 €	24	6	0 €
Licencia Jupyter Notebook	0 €	24	6	0 €
Total				**134,75 €**

Debido a que la mayoría de los programas utilizados son de código abierto, el precio de los materiales no es demasiado alto. Además, la amortización abarata los costes, ya que la duración del proyecto no es demasiado larga.

6.3.3 Costes indirectos

En esta sección describiremos los costes indirectos asociados al desarrollo del presente proyecto. Podemos definir los costes indirectos como los gastos relacionados con el gasto de luz, internet, teléfono, etc. Se ha decidido establecer un 5% sobre los gastos directos, como se puede observar en la Tabla 6.5:

Tabla 6.5: Costes indirectos

COSTE	IMPORTE
Coste de personal	9.250,62 €
Coste de material	134,75 €
Total costes directos	9.385,37 €
Costes Indirectos (5%)	**469,27 €**

6.3.4 Costes totales

En la siguiente sección realizaremos el cálculo del coste total del proyecto, estudiando los cálculos previos establecidos. Asumiremos el impuesto del valor añadido (IVA) incluido en todos los importes descritos hasta el momento. Hemos asumido añadir un 10% en concepto de beneficios, por lo tanto, en la Tabla 6.6 podemos ver los costes totales, que se estiman en DIEZ MIL OCHOCIENTOS CUARENTA CON DIEZ EUROS y obtener un beneficio de 985,46 euros.

Tabla 6.6: Costes totales

Descripción	Importe
Coste de personal	9.250,62 €
Coste de material	134,75 €
Costes indirectos (5%)	469,27 €
Importe total sin beneficios	9.854,64 €
Beneficio (10%)	985,46 €
Presupuesto total	10.840,10 €

7. Conclusiones y trabajos futuros

En este último capítulo se expondrán las conclusiones finales. Además, se explicarán las posibles mejoras y las líneas de investigación futuras que podrían aplicarse al proyecto.

7.1 Conclusiones

Una vez finalizado el proyecto, hemos logrado realizar una simulación de un ejercicio terapéutico contra la apraxia del habla. Con este simulacro, se pretende crear en un futuro un ejercicio que pueda ayudar a la gente aquejada de apraxia del habla, que es una consecuencia habitual en ancianos que sufran Alzhéimer, pacientes con enfermedades neurodegenerativas o que hayan sufrido un Ictus en algún momento de su vida.

El ejercicio de la terapia está completamente orientado a su uso con el robot Mini. Este robot es el encargado de interactuar con el paciente y tiene el objetivo de enseñar al usuario cómo pronunciar las vocales. Al utilizar imágenes en tiempo real para el desarrollo de la actividad, ésta se vuelve muy dinámica, debido a que la respuesta frente a las acciones del usuario es rápida. Gracias a esto, se puede realizar la corrección en el mismo instante en el que se realiza el fallo de pronunciación. Además, al utilizar un robot intentamos hacer el ejercicio más interesante para el usuario, lo que refuerza la terapia y favorece la progresión del paciente.

Se ha obtenido una gran cantidad de información con la realización del presente proyecto. Sin ninguna noción previa, se ha conseguido aprender a utilizar el framework ROS, lo que es indispensable para proyectos de esta índole. Gracias a este framework, hemos podido comunicar los nodos de nuestro sistema mediante la comunicación por mensajes que nos ofrece, además de darnos la arquitectura de paquetes y nodos, con la que podemos hacer realidad nuestro sistema. Podemos añadir que ROS nos ha dado algunas herramientas y rutinas que nos han facilitado en gran medida el trabajo.

Otro campo donde también se ha profundizado es sobre el Machine Learning, o Aprendizaje Automático. Esta rama está en auge dentro de la ingeniería del software, y su aprendizaje resulta muy positivo. El núcleo de nuestro trabajo es la creación de un clasificador que nos ayude a predecir qué vocal está pronunciando el paciente, lo que nos hace posible corregirle en el caso de producirse un fallo. Debido a esto, es completamente indispensable la utilización del Aprendizaje Automático.

Para concluir, nuestro objetivo principal era la creación de un ejercicio que pudiese ser tomado como punto de contacto para la creación de terapias más complejas utilizando el robot Mini, y podemos decir que ha sido logrado exitosamente.

7.2 Líneas futuras

Una vez finalizado el proyecto, tenemos la capacidad de estudiar los sistemas y elementos que conforman nuestro sistema, lo que nos da una visión de las líneas de investigación futuras que se abren.

Este proyecto únicamente trabaja con poses vocales pronunciadas por el usuario. Una línea de investigación podría ser ampliar las capacidades del sistema, pudiendo reconocer monosílabos o incluso palabras enteras. Esto daría un salto de complejidad al usuario al realizar la terapia, ya que los fonemas propuestos no serían tan sencillos como en el caso de pronunciar las vocales. Para ello, se debería de volver a estudiar el apartado de la detección de la boca, ya que quizá se necesitaría utilizar más partes del rostro, como puede ser el mentón o las mejillas, para captar mejor lo que dice el usuario.

Debido a las limitaciones que nos ha impuesto el detector facial utilizado (ver Sección 4.1.3), no se han conseguido resultados satisfactorios con más de 3 poses vocales. Como trabajo futuro se propone la integración de otras opciones de reconocimiento de rostro que permitan la detección de más poses, ya que el software Stasm está pensado para trabajar con rostros serios. Con un programa que trabaje adecuadamente bocas abiertas, se podrían implementar las cinco vocales e incluso otros tipos de poses vocales.

También, cabe destacar que se podría estudiar una mejora en las interacciones con el usuario. Una buena interacción usuario-robot impulsa una mejora en los resultados de la terapia, ya que motiva al usuario y hace que éste no pierda interés por el ejercicio. Por ello, se podrían diseñar mejores interacciones orientadas a un tipo concreto de pacientes, ya que no se puede actuar de igual manera frente a un anciano que frente a un niño. Este punto está pensado de cara a realizar pruebas con pacientes reales.

Por último, en este proyecto se realiza una propuesta de lo que podría ser un ejercicio para la terapia contra la apraxia del habla. Este ejercicio no está diseñado por ningún especialista, por lo que, si se desea llegar a tener unos resultados realmente adecuados de cara a los pacientes, se debería diseñar un ejercicio bajo la supervisión de un logopeda o un especialista en la materia.

Bibliografía

[1] R. González Victoriano y L. Toledo Rodríguez, «A Apraxia del Habla: Evaluación y Tratamiento,» 2015. [En línea]. Available: http://repositorio.uchile.cl/handle/2250/134234. [Último acceso: 05 09 2016].

[2] A. Ygual Fernández y J. F. Cervera Mérida, «Dispraxia verbal: características clínicas y tratamiento logopédico,» *Neurol,* vol. 40, nº 1, pp. 121-126, 2005.

[3] A. Costa, G. Asin-Prieto, S. Shimoda, E. Iánez, J. C. Moreno, J. L. Pons y J. M. Azorin, «INTEGRACION DE INTERFAZ CEREBRO-COMPUTADOR Y EXOESQUELETO DE MIEMBRO INFERIOR ORIENTADO A LA REHABILITACION.,» de *Actas de las XXXVI Jornadas de Automática*, Bilbao, 2015.

[4] B. Gallardo, C. Hernández y V. Moreno, «Lingüística clínica y neuropsicología cognitiva,» de *Actas del Primer Congreso Nacional de Lingüística Clínica.* , Valencia.

[5] O. J. Radabán y A. P. Rozas, «PROBLEMAS DEL LENGUAJE EN LA TERCERA EDAD. ORIENTACIONES Y PERSPECTIVAS DE LA LOGOPEDIA,» *Revista galego-portuguesa de psicoloxía e educación: revista de estudios e investigación en psicología y educación,* nº 8, pp. 387-398, 2002.

[6] M. T. Aguilar, H. O. Rodriguez, J. N. Ravelo, M. C. C. Moinelo, T. F. González, E. Q. Rodríguez y E. F. Martínez, «Actividades para la corrección de la apraxia constructiva en pacientes con secuelas de enfermedad cerebro-vascular,» *Revista electrónica de terapia ocupacional,* nº 8, 2008.

[7] M. Mercadal y P. Martí, «Aplicación de la musicoterapia en las demencias. Informaciones psiquiátricas,» *Publicación científica de los Centros de la Congregación de Hermanas Hospitalarias del Sagrado Corazón de Jesús,* nº 188, pp. 119-126, 2007.

[8] O. Sacks y C. Tomaino, «Music and neurological disorder,» *International Journal of Arts Medicine,* vol. 1, nº 1, pp. 10-12, 1991.

[9] T. Shibata, K. Wada y K. Tanie, «Subjective evaluation of seal robot at the Japan Cultural Institute in Rome,» de *ICCAS*, 2003.

[10] K. Wada, Y. Ikeda, K. Inoue y R. Uehara, «Development and preliminary evaluation of a caregiver's manual for robot therapy using the therapeutic seal robot Paro,» de *19th International Symposium in Robot and Human Interactive Communication*, Viareggio, 2010.

[11] Y. Furuta, M. Kanoh, T. Shimizu, M. Shimizu y T. Nakamura, «Subjective evaluation of use of Babyloid for doll therapy,» de *In Fuzzy Systems (FUZZ-IEEE), 2012 IEEE International Conference on*, 2012.

[12] L. F. Cossio, J. M. L. Salvador y S. F. Martínez, «Robotics for social welfare,» *Journal of accessibility and design for all,* vol. 2, nº 1, pp. 94-113, 2012.

[13] R. S. Michalski, J. G. Carbonell y T. M. Mitchell, Machine Learning: An Artificial Intelligence Approach, Springer Science & Business Media, 2013.

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot y E. Duchesnay, «Scikit-learn: Machine Learning in Python,» *Journal of Machine Learning Research,* vol. 12, pp. 2825--2830, 2011.

[15] M. Salichs, E. Salichs, I. Encinar, A. Castro-González y M. Malfaz, «Estudio de escenarios de uso para un robot social asistencial para enfermos de Alzheimer,» de *Actas de las XXXV Jornadas de Automática*, Valencia, 2014.

[16] Z. Zhang, «Microsoft kinect sensor and its effect,» *IEEE Multimedia,* vol. 19, nº 2, pp. 4-10, 2012.

[17] M. A. Salichs, R. Barber, A. M. Khamis, M. Malfaz, J. F. Gorostiza, R. Pacheco, R. Rivas, A. Corrales, E. Delgado y D. Garcia, «Maggie: A robotic platform for human-robot social interaction.,» de *IEEE Conference on Robotics, Automation and Mechatronics*, 2006.

[18] B. Leonardo, B. Claudia y Q. Silvia, «A general approach to TTS reading of mixed-language texts.,» de *In Proc. of 5th ISCA tutorial and research workshop on speech synthesis.*, 2004.

[19] J. Bohren y S. Cousins, «The SMACH high-level executive [ROS news],» *IEEE Robotics & Automation Magazine,* vol. 17, nº 4, pp. 18-20, 2010.

[20] M. Quigley, B. Gerkey, K. Conley, J. Faus, T. Foote, J. Leibs, E. Berger, R. Wheeler y A. Ng, «ROS: an open-source Robot Operating System,» *En ICRA workshop on open source software,* vol. 3, nº 3.2, 2009.

[21] S. Sonnenburg, G. Raetsch, S. Henschel, C. Widmer, J. Behr, A. Zien, F. de Bona, A. Binder, C. Gehl y V. Franc, «The SHOGUN Machine Learning Toolbox.,» *Journal of Machine Learning Research,* nº 11, pp. 1799-1802, Junio 2010.

[22] T. Smith y E. Frank, «Statistical Genomics: Methods and Protocols,» de *Introducing Machine Learning Concepts with WEKA*, New York, Springer, 2016, pp. 353-378.

[23] J. Guevara-Díaz, *Deteccion de Rostros por medio de las Wavelets de Morlet,* Trujillo: Universidad Nacional de Trujillo, Technical reports.

[24] S. M. a. F. Nicolls, «Active Shape Models with SIFT Descriptors and MARS,» *VISAPP,* 2014.

[25] T. Cootes, C. Taylor, D. Cooper y J. Graham, «Active shape models - their training and application,» *Computer Vision and Image Understanding,* nº 61, pp. 38-59, 1995.

[26] «Anaconda Software Distribution. Computer software,» Nov 2016. [En línea]. Available: https://continuum.io. [Último acceso: 06 Sept 2016].

[27] F. Pérez y B. Granger, «IPython: A System for Interactive Scientific Computing,» *Computing in Science and Engineering,* vol. 9, nº 3, pp. 21-29, May 2007.

[28] «Convenio colectivo de ámbito estatal para los centros de educación universitaria e investigación,» [En línea]. Available: https://www.boe.es/boe/dias/2015/02/11/pdfs/BOE-A-2015-1343.pdf. [Último acceso: 10 Sept 2016].

[29] Y. Furuta, M. Kanoh, T. Shimizu, M. Shimizu y T. Nakamura, «Subjective evaluation of use of Babyloid for doll therapy,» de *Fuzzy Systems (FUZZ-IEEE), 2012 IEEE International Conference on*, Brisbane, QLD, 2012.

Apéndice A: Manual de instalación

En este apéndice detallaremos el manual de instalación de los componentes necesarios para la realización del proyecto. Estos manuales de instalación han sido desarrollados para el sistema operativo Ubuntu 14.04.

A.1 Instalación del framework ROS

En el presente proyecto es necesaria la instalación del framework ROS en su versión Índigo, que es la aceptada por el robot Mini. A continuación, describiremos la guía de instalación de dicho framework:

1. Configurar los repositorios de Ubuntu: se deben configurar los repositorios para permitir aquellos que sean “restricted”, “universe” y “multiverse”.

2. Configurar el sources.list: se debe configurar el ordenador para que acepte software proveniente de packages.ros.org. Esto se debe realizar mediante el siguiente comando:

   $ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'

3. Configurar las claves necesarias para la instalación de ROS:

   $ sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net --recv-key 0xB01FA116

4. Instalación:

   4.1 Asegurarse de que el índice de paquetes está actualizado:

   $ sudo apt-get update

   4.2 Realizar la instalación:

   $ sudo apt-get install ros-indigo-desktop-full

5. Inicializar rosdep: esto nos permitirá instalar fácilmente dependencias del sistema necesarias para utilizar determinadas funcionalidades de ROS:

   $ sudo rosdep init
   $ rosdep update

6. Configurar el entorno:

   $ echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc
   $ source ~/.bashrc

7. Obtener rosinstall: esta herramienta nos permitirá descargar código fuente de los paquetes de ROS con un único comando:

   $ sudo apt-get install python-rosinstall

A.2 Instalación Scikit-Learn

Scikit-Learn es el software que contiene las librerías necesarias para crear, entrenar y utilizar el clasificador que utilizaremos en el presente proyecto. El procedimiento para instalar dicho software es el siguiente:

1. Instalación del instalador anaconda: este instalador, aparte de permitirnos instalar las librerías de Scikit-Learn, viene con las herramientas numpy y scypy que son necesarias para el funcionamiento de dichas librerías. Los pasos para realizar la instalación son:

   1.1 Descargar el instalador desde la siguiente página web: “https://www.continuum.io/downloads”

   1.2 Dependiendo de la versión de Python que se utilice, introducir la siguiente rutina y seguir las instrucciones que nos muestre:

   Para Python 3.5: *$ bash Anaconda3-4.1.1-Linux-x86_64.sh*
   Para Python 2.7: *$ bash Anaconda2-4.1.1-Linux-x86_64.sh*

2. Instalación de Scikit-Learn:

   conda install scikit-learn

A.3 Instalación Stasm

Para el funcionamiento del nodo Detection, se debe de instalar el software Stasm, que es el encargado de caracterizar el rostro del usuario con los puntos que luego tomaremos para realizar el presente proyecto. Para su correcta instalación, se deben de seguir los siguientes pasos:

1. Descargar el software:

   $ wget http://www.milbo.org/stasm−files/4/stasm4.1.0.tar.gz

2. Descomprimir la carpeta obtenida para obtener la carpeta “stasm4.1.0”

3. Descargar de la siguiente url: “https://github.com/juan-cardelino/stasm” los archivos appmisc.cpp.20140201.diff y shapefile.cpp.20140201.diff y copiarlos en la carpeta que les corresponda.

4. Parchear los siguientes archivos utilizando estos comandos:

   $ cd stasm4.1.0/apps

   $ patch −p0 < appmisc.cpp.20140201.diff

   $ cd stasm4.1.0/apps/shapefile

   $ patch −p0 < shapefile.cpp.20140201.diff

5. Escribir el comando cmake para realizar la construcción de la librería.

6. Modificar los directorios del archivo CMakeCache para adecuarlo a nuestro sistema

7. Ejecutar comando make.

Apéndice B: Manual de uso del sistema

B.1 Manual de entrenamiento del clasificador

Antes de poder iniciar el ejercicio diseñado, se debe entrenar nuestro clasificador para poder realizar estimaciones con los datos que nos lleguen del sensor Kinect. Se deben de seguir los siguientes pasos:

1. Grabación de los bags: se utilizará como entrada de datos en este apartado bags con las imágenes necesarias para el buen entrenamiento del clasificador. Para grabar se debe:

    1.1. Conectar el sensor Kinect al computador
    1.2. Introducir el siguiente comando:

    $ roslaunch openni_launch openni.launch

    1.3. Lanzar rviz para comprobar que el rostro a grabar se ve correctamente utilizando:

    $ rosrun rviz rviz

    1.4. Una vez situado correctamente el sujeto, grabar el bag utilizando:

    $ rosbag record -O [Introducir nombre] /camera/depth/image_raw /camera/depth/points /camera/rgb/image_color

2. Entrenamiento del clasificador:

    2.1. Iniciar la toma de datos a partir de:
    - **Bag (entrenamiento off-line): $** *rosbag play -l [Introducir nombre]*
    - **Kinect (entrenamiento on-line): $** *roslaunch openni_launch openni.launch*

2.2. Iniciar el nodo Detection:

$ rosrun vowel_identification_skill Detection

2.3. Iniciar el nodo de entrenamiento:

$ rosrun vowel_identification_skill Main_Train_Estimator

2.4. Seguir los pasos indicados en el nodo de entrenamiento.

B.2 Manual para arrancar el ejercicio

1. Arrancar el sensor Kinect:

   $ roslaunch openni_launch openni.launch

2. Iniciar el robot social Mini:

   $ roslaunch mini_bringup alz_start_basic.launch

3. Iniciar el nodo Detection:

   $ rosrun vowel_identification_skill Detection

4. Iniciar el nodo de la terapia:

   $ rosrun vowel_identification_skill Main_Game