Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERÍAS INDUSTRIALES

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TRABAJO FIN DE GRADO

DESARROLLO DE UN AVATAR VIRTUAL PARA SU INTEGRACIÓN EN UNA CABEZA ROBÓTICA

Autor:

Álvarez Pérez, Daniel

Tutor(es):

Gómez García-Bermejo , Jaime

Ingeniería de Sistemas y Automática

Valladolid, Enero 2020.

AGRADECIMIENTOS

Quería expresar mi agradecimiento a los profesores Jaime Gómez García-Bermejo y Eduardo Zalama Casanova por servir de guía durante el transcurso del trabajo. De igual manera agradezco al profesor David Loza Matovelle, de la Universidad Fuerzas Armadas ESPE de Ecuador, su ayuda durante la realización de algunas partes del mismo.

Resumen y Palabras Clave

El presente trabajo Fin de Grado consiste en el diseño y creación de un rostro de naturaleza virtual para su implementación en una cabeza robótica. El sistema para crear el rostro se basa en la retroproyección de un avatar virtual 3D sobre una máscara translúcida con ayuda de un mini proyector más una lente, ambos ubicados detrás de la máscara. La máscara sirve como pantalla física para proyectar la imagen del avatar. El avatar virtual es el que da la apariencia y los rasgos faciales al rostro del robot. Dicho avatar también estará animado y compuesto por un sistema de acción para simular el comportamiento gestual del ser humano. Este trabajo sirve como plataforma para el estudio de nuevas formas de comunicación visual entre el hombre y robot.

Palabras clave: *Retroproyección, avatar virtual, unidad de acción, animación facial, visema.*

Abstract and Keywords

The present Final Degree Project consists in the design and creation of a virtual nature face for its implementation in a robotic head. The system for creating the face is based on the rear projection of a 3D virtual avatar on a translucent mask, with the help of a mini projector plus a lens, both located behind the mask. The mask serves as a physical screen to project the avatar image. The virtual avatar is what gives the appearance and facial features to the robot's face. This avatar will also be animated and composed of an action system to simulate the gestual behaviour of the human being. This work serves as a platform for the study of new forms of visual communication between human and robot.

Keywords: *Rear projection, virtual avatar, action unit, face animation, viseme.*

Índice General

Índice de figuras

Índice de tablas

Capítulo 1 – Introducción

1.1. Marco del proyecto

El presente Trabajo Fin de Grado "Desarrollo de un avatar virtual para su integración en una cabeza robótica" propone el diseño y la implementación de un rostro robótico basado en la retroproyección de un avatar virtual sobre una cabeza robótica. Este trabajo forma parte de un proyecto de creación de un robot social cuyo objetivo es entablar comunicación con los seres humanos o reaccionar ante estímulos externos. Dentro de los sistemas de la comunicación, uno de los más importantes aparte de la comunicación verbal, también es la comunicación gestual que se percibe a través de la vista.

El funcionamiento general de un sistema robótico se basa en los procesos de percepción, control y actuación. Mediante sensores, el robot percibe y capta la información exterior. Un sistema de control recibe los datos percibidos y decide actuar. Por último la acción se lleva a cabo por el actuador. En el presente trabajo se implementará el medio o soporte de actuación del robot para la comunicación gestual, que es el rostro junto a sus "músculos faciales".

Específicamente, en un rostro se pueden considerar dos características principales, la apariencia y el comportamiento. La apariencia es la forma o el aspecto visual que tendrá el rostro. Mientras que el comportamiento es su modo de actuar. En este caso la apariencia del rostro será el de un avatar virtual 3D diseñado en un programa de modelado. El comportamiento vendrá dado a través de las animaciones de dicho avatar, mediante la creación de un conjunto de unidades de acción independientes entre sí pero que pueden combinarse para formar distintos movimientos faciales.

La representación del rostro en el robot se realizará mediante retroproyección. La retroproyección consiste en la proyección trasera de una imagen, en este caso la imagen del avatar, sobre una pantalla que será una máscara translucida, de forma que la persona situada delante de la máscara visualice la imagen proyectada a través de ella. Los componentes físicos empleados para este propósito son una máscara translúcida y un mini proyector con una lente ojo de pez montado detrás de la máscara, en el interior de la cabeza del robot. El avatar ha sido realizado con ayuda del programa de modelado Blender. Para realizar el control de las animaciones faciales del avatar se ha empleado el sistema operativo robótico ROS.

Actualmente son más conocidos los rostros robóticos de naturaleza electromecánica, pero también está la alternativa de rostros basados en modelos virtuales. Los rostros electromecánicos proveen una mayor sensación de presencia física, pero los segundos también tienen ventajas. En concreto, y en referencia a la cuestión de por qué implementar en este trabajo un rostro basado en un avatar virtual y no un rostro de naturaleza electromecánica se expone en los siguientes puntos:

- *Costes y componentes*: Los rostros de naturaleza electromecánica llevan implementados actuadores para simular el movimiento de los músculos. Evidentemente para este tipo de robots hay que añadir los costes de componentes como servomotores, rodamientos y aparte elementos estéticos que den la apariencia de un rostro humano. Otro coste a mayores también seria, si así se desea, añadir una capa que ejerza la función de la piel, para proteger los elementos de control y actuación internos del rostro y darle un aspecto más estético y realista. Por otra parte, los rostros basados en avatares, requieren de menos componentes y por tanto menos costes generalmente. Para este tipo de rostros se incluyen los costes del programa de modelado 3D, si bien en algunos casos estos programas son gratuitos, el coste del proyector y el coste de fabricación de la máscara que actúe como pantalla de proyección.
- *Mantenimiento*: En los robots electromecánicos hay que hacer inspecciones periódicas para detectar y corregir el desgaste de motores y demás componentes, ya que a todos los elementos mecánicos les afecta el paso del tiempo. En el caso de rostros basado en avatares no se requiere de un mantenimiento periódico.
- *Flexibilidad*: Un modelo avatar normalmente resulta más flexible para realizar cambios en el diseño. Estos modelos computacionales, al estar formados por una malla poligonal 3D, cualquier punto de la malla es editable y sujeto a cambios de posición. También para un mismo modelo o malla tridimensional se pueden añadir distintas texturas superficiales. Y respecto a la proyección, para una misma máscara genérica se pueden llegar a proyectar distintos rostros de avatares de proporciones similares.
- *Expresividad*: Al igual que los robots electromecánicos pueden adquirir una mayor sensación de presencia física, la expresividad facial en los avatares se puede conseguir más eficientemente. En estos casos, el movimiento del rostro depende únicamente de la programación que el usuario ha realizado sobre el modelo virtual tridimensional, y este movimiento se puede ajustar sin necesidad de verse supeditado a restricciones electromecánicas. Además el ajuste puede mantenerse fijo salvo que el usuario lo modifique. En los rostros de naturaleza electromecánica pueden existir restricciones y desajustes del movimiento, ya que estos movimientos no solo dependen de la programación realizada por el usuario, sino del estado de los componentes electromecánicos.

Por último cabe mencionar que en este trabajo está incluida la búsqueda bibliográfica sobre el uso e implementación de avatares virtuales en robots, el desarrollo desde cero de un avatar virtual, la implementación y el control de expresiones faciales sobre el avatar diseñado, un sistema de voz y sincronía labial y la creación de la máscara translúcida. El avatar virtual finalmente se proyectará sobre esta máscara para simular un rostro humano con sus correspondientes rasgos y animaciones faciales.

1.2. Objetivos del proyecto

El objetivo principal del presente trabajo Fin de Grado es el desarrollo de un avatar virtual tridimensional de aspecto humano, y su posterior integración en una cabeza robótica con el fin de crear el rostro de un robot basado en un modelo virtual. Este proyecto se realiza en concurrencia con otro Trabajo Fin de Grado titulado "Desarrollo de una cabeza robótica con alta capacidad gestual" que consiste en la creación de la cabeza y cuello del robot, y la programación de los distintos movimientos de giro y cabeceo.

El objetivo principal mencionado es una tarea de gran alcance. Para organizar, estructurar y facilitar su obtención, este objetivo se ha dividido en varios sub-objetivos que permitirán enfocar el trabajo en tareas de menor alcance y más específicas. Así, la consecución de todos ellos derivará finalmente en la consecución del objetivo principal. A continuación se citan los distintos sub-objetivos que a su vez conformarán las distintas fases del trabajo:

- Desarrollo del avatar virtual: La primera tarea consistirá en el desarrollo del avatar virtual 3D. Los tres procesos que constituyen el desarrollo del avatar son:
  - Modelado: Proceso para generar la forma del avatar 3D. Esta forma 3D consistirá en una malla tridimensional poligonal. Se buscará que el avatar tenga aspecto antropomórfico en apariencia y proporciones. El modelo no será de cuerpo entero, solo incluirá la cabeza y el cuello (aunque para la proyección sobre la máscara solo se emplee la zona del rostro).
  - Texturizado: Proceso para añadir texturas sobre la superficie de la malla 3D del modelo. La textura es el patrón que romperá con la apariencia uniforme de la superficie de la malla y dará color a los distintos elementos del rostro.
  - Animación: Proceso para crear los distintos gestos y movimientos faciales del avatar, que después se ejecutarán empleando el motor de juego de Blender. Al igual que la apariencia antropomórfica en el proceso de modelado, en la animación se intentará obtener un comportamiento gestual humano de las expresiones faciales, incluyendo las principales expresiones emocionales.

- Control automático de las animaciones faciales: Esta tarea consistirá en implementar un control automático de los distintos movimientos faciales de forma externa a Blender. Para ello se va a emplear un mecanismo que sea capaz de enviar las distintas unidades de movimientos al modelo, de modo que este pueda activar los actuadores correspondientes de forma automática. Los dos puntos principales que incluyen esta tarea son:
  - Comunicación ROS y Python con Blender: Se emplearán nodos de ROS y Python para enviar la información a Blender.
  - Gestor de comportamiento: Se desarrollará una rutina de expresiones faciales que realizará el avatar.
- Implementación del sistema avatar-máscara: El sistema avatar-máscara es el sistema que incluye los distintos componentes para formar el rostro del robot, incluyendo el sistema de proyección empleado. El rostro del robot se basa en la retroproyección de la imagen del avatar diseñado sobre una máscara translúcida que se acoplará en la cabeza del robot. Los procesos que incluyen esta tarea son los siguientes:
  - Diseño y fabricación de la máscara translúcida.
  - Elección del proyector y búsqueda de un método de proyección óptimo.
- Sistema del Voz y sincronía labial en tiempo real: Por último, como complemento se avanzará hacia la implementación de voz en el avatar junto a un sistema de sincronía gestual mediante la ejecución de visemas del lenguaje oral, de modo que el avatar pueda gesticular a la vez que hable.

1.3. Estructura de la memoria

La memoria consta de varios capítulos estructurados conforme al orden seguido en la realización del proyecto.

- *CAPÍTULO 1 - INTRODUCCIÓN:* En este primer capítulo se presenta el tema del proyecto y los objetivos principales del mismo.

- *CAPÍTULO 2 - ANÁLISIS Y ESTUDIO TEÓRICO DEL PROYECTO*: Se engloba todo el ámbito teórico del trabajo. Se incluye tanto la justificación como aplicaciones prácticas. Asimismo, se expone el estudio de otros proyectos previos relacionados con la tarea actual. Por último, se explica el sistema de codificación facial que se implementará en el control del avatar en capítulos posteriores.

- *CAPÍTULO 3 - CONSTRUCCIÓN DEL AVATAR*: En este capítulo se expone el modelado, diseño y animación del avatar.

- *CAPÍTULO 4 - CONTROL DE LAS ANIMACIONES MEDIANTE ROS:* Capítulo dedicado al control de los movimientos faciales del avatar.

- *CAPÍTULO 5 - VOZ Y SINCRONIZACIÓN LABIAL EN TIEMPO REAL:* Capítulo en el que se desarrolla el sistema de habla, empleando un convertidor texto a voz y la sincronía labial del avatar en tiempo real.

- *CAPÍTULO 6 - IMPLEMENTACION DEL SISTEMA AVATAR-MÁSCARA*: Construcción de la máscara y el sistema empleado para que la imagen del avatar se proyecte sobre la máscara.

- *CAPÍTULO 7 - EXPERIMENTACIÓN Y ANÁLISIS DE RESULTADOS:* Capítulo que presenta los resultados obtenidos en el trabajo.

- *CAPÍTULO 8 - ESTUDIO ECONÓMICO:* Capítulo dedicado al cálculo de los costes del proyecto.

- *CAPÍTULO 9 - CONCLUSIONES Y LÍNEAS FUTURAS:* Conclusiones extraídas del trabajo realizado y posibles líneas de trabajo futuras.

- *REFERENCIAS BIBLIOGRÁFICAS*

- *ANEXOS*

Capítulo 2 – Análisis y estudio teórico del proyecto

2.1. Introducción

El presente proyecto está enmarcado dentro del campo de la interacción Hombre-Máquina (HMI). Esta disciplina está dedicada a la búsqueda de sistemas capaces de establecer una comunicación entre computadoras/procesos y el ser humano, donde se intercambie información entre humano y máquina. En el campo de la robótica hay un interés creciente en el desarrollo de dispositivos que incluyan una funcionalidad social y métodos de interacción similares a los dados por la comunicación entre seres humanos. Esto incluye robots con altas capacidades de expresividad y/o equipados con avanzadas interfaces de comunicación en base a un lenguaje natural, para así poder integrarse en la comunicación con las personas. Un sistema adecuado para implementar las capacidades de expresividad y gestualidad en un robot, es mediante los avatares virtuales.

2.2. Justificación y aplicaciones prácticas

El ser humano a lo largo de su historia ha requerido de la necesidad de asignar una identidad a cualquier objeto o ser vivo como medio de distinción para explicar su funcionamiento, o establecer una comunicación con él. Esta forma de identificación era y es empleando un nombre, y asociando este nombre al aspecto físico del ser o persona correspondiente.

En la actualidad, en la era digital, se han establecido nuevos patrones de comunicación. No solo existe la conexión digital de objetos cotidianos con Internet, sino que las personas también están interconectadas a través de la red. Es por eso que dentro del mundo virtual, además de requerir un nombre identificativo, ha surgido la presencia de los avatares virtuales para representar el aspecto físico del usuario con el que establecer la comunicación virtual.

Un avatar virtual es un modelo o representación no real, que trata de personalizar o simbolizar la figura de un usuario al cual el avatar está asociado. Un avatar puede estar diseñado en 2D o 3D. En la actualidad, el empleo de los avatares ya es muy abundante en muchas aplicaciones, todos ellos vinculados a la comunicación ente un emisor y un

receptor. Y con los años va a ir incrementando más su uso conforme continúe la evolución de los sistemas de comunicación. A continuación, se muestran algunos ejemplos de la aplicación de avatares virtuales en varios campos:

- Entretenimiento: A principios de siglo XXI con el emerger de las primeras redes sociales, el uso de los avatares se estandarizó como método de reconocimiento entre los usuarios. El nombre de usuario y el avatar formaban los dos pilares de la identidad de los internautas. Sin embargo, el uso de avatares viene de antes, de finales del siglo XX a través de la industria de los videojuegos. El concepto de avatar surgió principalmente en los juegos en línea donde se requería la cooperación y la comunicación de los participantes. Los primeros mundos virtuales eran comunidades de chat donde solo se requería de un nombre de usuario. Pero en 1985, se desarrolló uno de los primeros mundos virtuales que fue un entorno gráfico en 2D llamado Habitat [1]. Los jugadores podían elegir un avatar, y con este avatar establecer distintas acciones con otros avatares de otros usuarios dentro del mundo virtual. Con el paso de los años, los avatares se han ido perfeccionando, llegando a un punto realista de ser prácticamente idénticos a la persona o usuario que representan. Un ejemplo de este realismo, surgió con las consolas de última generación. A partir de una cámara en la consola, se captaban los puntos clave del rostro o cuerpo del usuario, y mediante visión artificial y técnicas de modelado potentes se creaba un avatar muy similar al jugador.
- Medicina: En la medicina se ha impulsado de manera notoria la presencia de avatares virtuales en distintos ámbitos para el tratamiento de la salud de los pacientes. Una de las últimas aplicaciones destacadas en este ámbito es el proyecto CicerOn [2]. Este proyecto nace para dar tratamiento a personas con autismo o síndrome de Asperger. Mediante la aplicación CicerOn y con el uso de unas gafas VR, los pacientes se introducen en una realidad virtual en donde residen avatares. Los pacientes tienen que interactuar con los avatares virtuales para entablar comunicación y superar retos. La finalidad es que los pacientes puedan perder la ansiedad, el rechazo y el miedo que les provoca vivir esas situaciones en la realidad. Otro de los servicios donde los avatares están destacando, es en la asistencia sanitaria. Uno de los ejemplos más notorios es el proyecto Kristina [3] [4], que se terminó de desarrollar en 2018. En el proyecto se proponía crear un avatar virtual que actuase de asistente sanitario como sistema para reducir la congestión de las consultas sanitarias de atención primaria a inmigrantes (figura 2.1). Para realizar esta función, el avatar era capaz de entablar conversación en cinco idiomas. El objetivo de dicho proyecto es ofrecer una solución tecnológica para facilitar la superación de las barreras culturales y lingüísticas en los servicios de atención sanitaria de personas inmigrantes.

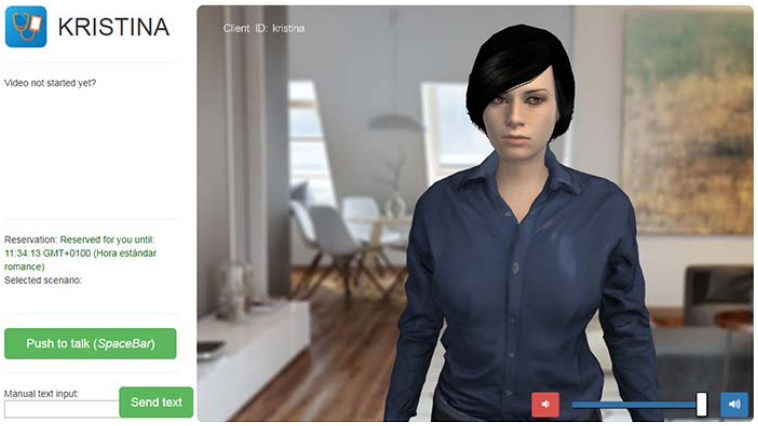


Figura 2.1 - Agente virtual Kristina desarrollado en el proyecto "Kristina"
Imagen extraída de la página web oficial del proyecto: http://kristina-project.eu/

- Educación: En el ámbito de la educación también se ha encontrado utilidad para los avatares virtuales. Una de las aplicaciones más interesantes ha sido el uso de los avatares como método de entrenamiento para los docentes [5]. Esta idea fue desarrollada en un proyecto llevado a cabo por la Universidad Central de Florida. El proyecto consiste en la creación de un simulador de una clase real, llamado TeachLive. El profesor se sitúa delante de una pantalla donde se proyecta la interfaz con aspecto de aula. En ella se muestran distintos avatares que representan a los estudiantes, cada avatar simula el comportamiento de distintos tipos de alumnos. La finalidad es ayudar al profesor en la forma de dar las clases, tanto en la forma de explicar cómo en la forma de establecer la relación con el alumnado. Estos avatares están entrenados para percibir y transmitir lo que un alumno puede llegar a captar por parte de un docente.

Figura 2.2 - Avatares virtuales dentro del simulador TeachLive.
Imagen extraída de la página web oficial del proyecto: http://teachlive.org/

- Robótica: En el marco de la robótica, los avatares virtuales están teniendo un gran impacto en varios ámbitos. Los avatares pueden servir como herramienta de diseño para simular los distintos movimientos que pueda realizar un robot. Otra aplicación incluye el servir como agentes o interfaces que interactúen con un robot dando órdenes o comunicándose con él. Y también pueden ser utilizados para ser implementados en el robot y darle un aspecto más antropomórfico. Un ejemplo seria el presente trabajo.

Este Trabajo Fin de Grado se ha elaborado con el fin de colaborar en la creación de un tipo de robot que pueda establecer comunicación con las personas. La utilidad específica de este trabajo dentro de todo lo que puede incluir el robot en su conjunto, es la de dar expresividad al rostro del robot mediante animaciones faciales realistas. Desde un punto de vista profesional, el uso de avatares virtuales como rostro puede servir en la creación de robots orientados a la telepresencia. Este tipo de robots se encargan de interactuar con el entorno y a la vez establecer una comunicación bidireccional con un usuario local que recibe o envía información a la máquina. Esto permite a la persona que controla de forma remota al robot, sentirse como si estuviera presente en un lugar que no es su ubicación real. El avatar virtual serviría como interfaz y rostro para este tipo de robots. A mayores, en estos robots también se incluyen componentes como micrófonos para captar el sonido externo, altavoces para reproducir sonido, cámara para que el usuario puede visualizar la ubicación remota, etc. Un ejemplo práctico de utilización sería para reuniones donde un empleado no pudiese asistir y el sustituto fuese un robot de telepresencia con un rostro basado en un modelo virtual. Otro ejemplo sería el uso de un robot/modelo virtual que pudiese servir de intermediario en la comunicación entre un doctor y un paciente de forma remota. Asimismo el robot también puede servir como interfaz primaria de un sistema de Inteligencia Artificial, por ejemplo para construir un asistente social autónomo que entable comunicación con las personas que se sitúen frente a él.

2.3. Retroproyección en cabezas robóticas

2.3.1. Introducción

El presente proyecto del avatar junto a la cabeza robótica, está basado en el concepto de robot social. Un robot social es un robot autónomo que interactúa y se comunica con los seres humanos u otros agentes físicos autónomos siguiendo los comportamientos sociales y las reglas asociadas a su función. El tema del robot social ha sido previamente tratado, estudiado y llevado a cabo por algunas grandes empresas dentro del mundo de la robótica. Como parte previa al desarrollo de este trabajo se ha considerado analizar la información existente acerca de proyectos previos, relacionados con robots sociales basados en la retroproyección de imágenes de avatares virtuales. Los proyectos más significativos que se han evaluado son los tratados en los apartados siguientes.

2.3.2. Mask-Bot

Mask-bot es un sistema robot que fue desarrollado para investigar nuevas metas en el campo de estudio HRI (Interacción Humano-Robot). Desde la comunicación visual del robot hasta el estudio de la síntesis de voz. Este proyecto fue llevado a cabo por el Instituto Técnico y de Sistemas Cognitivos de Múnich. La primera generación [6] de este sistema de robots estaba integrada por tres componentes principales: una máscara, un proyector LED y una unidad motora para posicionar cargas, controlado por ordenador.

El funcionamiento general consistía en la proyección de un avatar sobre la máscara traslucida para simular un rostro humano. Para el sistema de proyección, la idea era obtener una imagen proyectada del rostro, desde la barbilla hasta la frente. Para cubrir ese rango de proyección, el proyector tenía que posicionarse en una base inclinada, sobre un soporte. Con el fin de reducir la distancia de proyección y hacer el sistema más compacto, se empleó una lente ojo de pez. La lente ojo de pez ayuda a aumentar el ángulo de proyección.

La unidad motora de posicionamiento se colocó debajo de la máscara. Esta unidad realizaba la función de soporte de toda la carga, y también se encargaba de producir un movimiento vertical y horizontal del sistema para imitar el giro del cuello. Cada movimiento (vertical y horizontal) era realizado por un motor interno dentro de la unidad motora. Esta unidad estaba comunicada a un PC. Después, un programa de control enviaba información con la posición deseada en tiempo real. Para reducir los movimientos bruscos del cuello, se aplicó un control de velocidad y aceleración.

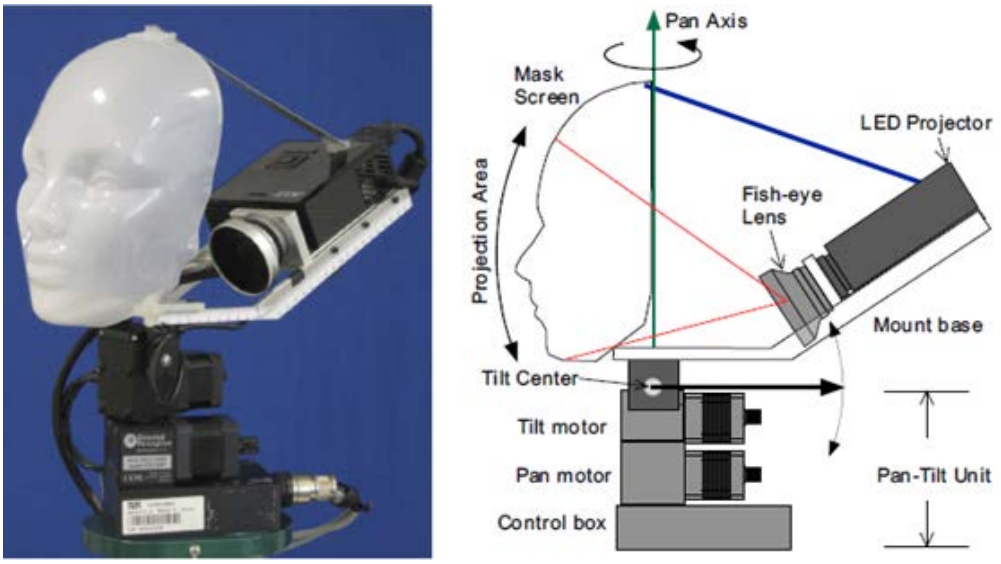


Figura 2.3- Imagen del Mask-bot de 1ºgeneración (izquierda) y un esquema de la configuración del mismo (derecha)
Imagen extraída del paper: Mask-bot"- a life-size talking head animated robot for AV speech and human-robot communication research. Realizado por: Takaaki Kuratate, Brenand Pierce, Gordon Cheng

A pesar de obtener resultados satisfactorios frente a una audiencia respecto al realismo de la cara, el *Mask-bot* de primera generación aun presentaba algunos inconvenientes:

- El rostro no era intercambiable y la propia mascara tenía sus propios rasgos. La máscara llevaba implementada la forma de la boca, los ojos y nariz. No se podían añadir distintos avatares y así experimentar con nuevos gestos faciales.
- Algunas animaciones del avatar como el movimiento de la mandíbula arriba y abajo, eran poco naturales. En parte debido al efecto producido por la máscara. Ya que la máscara llevaba implementadas las protuberancias de la boca, la nariz y los ojos.
- La unidad motora era bastante ruidosa, y el sistema solo tenía dos grados de libertad.
- En el apartado visual, al ser un proyector grande en dimensión, se veía la parte trasera del proyector y eso hacia perder el realismo que tuviera el rostro.

Este proyecto fue el primer prototipo creado por este Instituto Técnico de Múnich. Más adelante, se crearon nuevos modelos en donde implementaron nuevas mejoras. El siguiente prototipo creado fue el *Mask-bot 2i*, o el *Mask-bot* de segunda generación [7]. El núcleo de funcionamiento era el mismo. Un sistema basado en la proyección de un avatar virtual sobre una máscara. Pero en este caso se perfeccionaron muchos de los inconvenientes que presentaba la primera generación. Ahora el rostro era intercambiable. También se modificó el sistema de proyección para hacer el robot más compacto, y con un tamaño acorde a la dimensión de un rostro humano. Para ello, el proyector se colocó

en vertical de modo que ocupó menos espacio, y sobre él, se colocó un espejo en posición inclinada con un ángulo prefijado. La imagen proyectada se reflejaba sobre este espejo y llegaba a la máscara. Para encuadrar la imagen proyectada sobre la máscara, se implementó un brazo ajustable entre el soporte y la zona de la máscara para hacer pruebas experimentales hasta encajar correctamente la imagen. El brazo ajustable permitía modificar la posición y distancia de la máscara con respecto al soporte, que es donde se situaba el proyector y el espejo.

Asimismo, en la segunda generación se añadió también una cámara encima de la máscara para establecer un sistema de rastreo, en donde el robot pudiera hacer un tracking de la cara de las personas y detectar puntos clave.

El sistema de control del movimiento del cuello también mejoró con respecto al primer prototipo. Ahora la unidad de control estaba formada por dos niveles. Un nivel inferior formado por un módulo electrónico FPGA con tres codificadores rotatorios que se encargaban de codificar el movimiento mecánico de los tres motores. Este módulo también se encargaba de hacer un control PID para la velocidad. El nivel superior de control, se realizó con el entorno ROS desde un PC. Un nodo de ROS se utilizó como interfaz para comunicarse con el nivel inferior, y enviar como entradas la posición y velocidad deseadas.

Por último se cambió el aspecto de la máscara, disminuyendo las protuberancias de las zonas faciales, creando una máscara más genérica, pero a la vez más funcional para implementar distintos avatares.

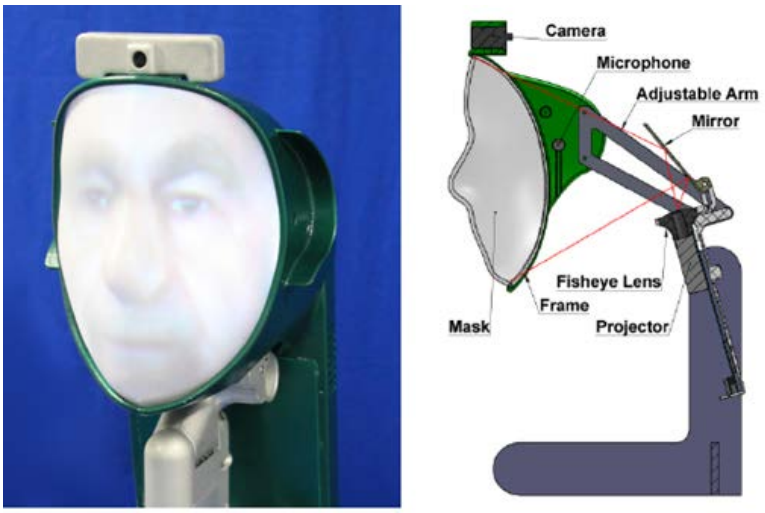


Figura 2.4- Imagen del Mask-bot de 2ºgeneración (izquierda) y un esquema de la configuración del mismo (derecha).
Imagen extraída del paper: "Mask-Bot 2i": An active customizable Robotic Head with Interchangeable Face. Realizado por: Brennand Pierce, Takaaki Kuratate, Christian Vogl and Gordon Cheng

2.3.3. Furhat Robot

Furhat robot es un robot social creado por la empresa *Furhat Robotics*. Este robot está considerado como uno de los robots sociales más avanzados hasta la fecha [8]. El robot es capaz de interactuar con los seres humanos, hablando, escuchando y también mostrando distintas emociones además de mantener el contacto visual con el usuario. Tiene la capacidad de hacer preguntas, realizar entrevistas, aprender habilidades y hablar en multitud de idiomas.

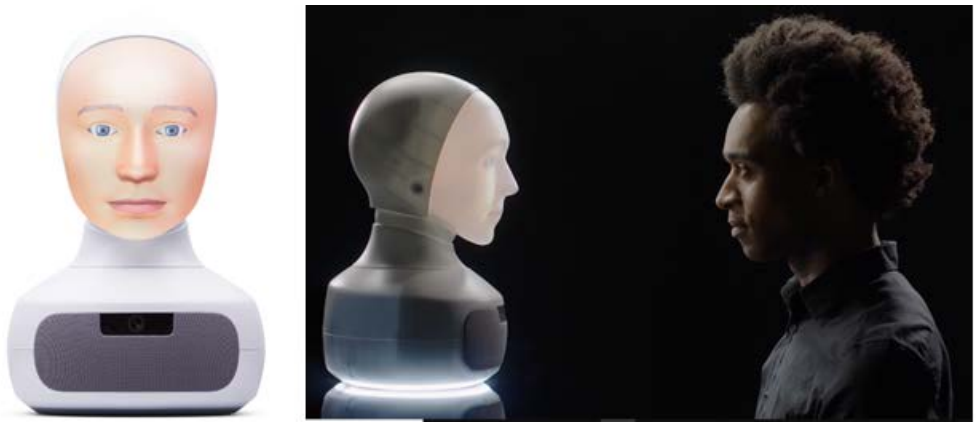


Figura 2.5- El robot Furhat del modelo actual.
Imagen extraída de la página web oficial de Furhat robotics: https://www.furhatrobotics.com/

Los primeros modelos [9] de estos robots tenían una configuración similar al *Mask-bot*. La zona de la cabeza estaba constituida por un sistema de proyección con un mini proyector, un espejo y una máscara. El proyector en este caso se situaba encima de la máscara proyectando la imagen hacia el espejo situado detrás de la máscara. Después la imagen se reflejaba en el espejo para quedar proyectada sobre la máscara. Para cubrir la parte superior y trasera de la cabeza donde se encontraba el proyector y el espejo se utilizaron gorros y sombreros. En cuanto al rostro, se decidió crear un modelo avatar en 3D por varias razones:

- En un avatar los labios del modelo facial se podían sincronizar automáticamente con una señal de habla que el sistema producía. Esto se realizó usando una transcripción de varios discursos hablados. El sistema que se empleó para la sincronización labial probó que el modelo creado demostró una mejora de la inteligibilidad del habla sobre la escucha de la señal de audio.
- El modelo del rostro del avatar ofrecía flexibilidad a la hora de controlar gestos y movimientos faciales como la mirada, parpados o cejas.

Figura 2.6- Los primeros modelos del robot Furhat con y sin sombrero.
Imagen extraída del paper: Furhat: A Back-projected Human-like Robot Head for Multiparty Human-Machine Interaction. Realizado por: Samer Al Moubayed, Jonas Beskow, Gabriel Skantze, Björn Granström

Para controlar el comportamiento de *Furhat* se empleó un sistema dirigido por eventos implementado en Java. Esto permitía al sistema reaccionar ante sucesos que eran las entradas externas como el habla o datos de proximidad, para después producir acciones como gestos faciales, movimiento de la cabeza o el habla. El diseñador de diálogo, se encargó de definir una jerarquía en el estado de los diálogos del robot y las acciones sensoriales emparejadas con esos estados. El dialogo del robot contenía dos estados mayores donde se reflejan distintas iniciativas: uno donde el robot *Furhat* llevaba la iniciativa de la conversación haciendo preguntas al usuario, y otro donde el usuario realizaba las preguntas a *Furhat*. El robot trataba de continuar el dialogo aunque en muchos casos tenía problemas para entender las respuestas del usuario, ya que el robot extraía principalmente las palabras clave. Cuando no había ninguna persona cerca de los sensores de proximidad, el robot bajaba la mirada. En cuanto alguien se acercaba a los sensores de proximidad, el robot miraba hacia la persona para iniciar una conversación.

Luego de la creación de los primeros modelos, el modelo *Furhat* adquirió notoriedad, y la empresa *Furhat Robotics* siguió creando modelos cada vez más avanzados. Se dejó a un lado el uso de gorros y sombreros para cubrir la cabeza para ir adquiriendo un aspecto más estético. También se mejoró la IA del sistema hasta llegar al modelo actual.

Finalizado el análisis de estos primeros modelos de *Furhat* y *Mask-bot*, se ha podido extraer alguna limitación de diseño que posteriormente se ha resuelto con el procedimiento empleado en el presente trabajo. Una de estas limitaciones reside en el sistema de proyección (tamaño del proyector, tamaño de espejo, distancia de proyección)

que es poco compacto en tamaño. Esto se observa tanto en el *Mask-bot*, donde se podía visualizar el proyector por la parte trasera, como en el *Furhat* donde se utilizaba un gorro para tapar el proyector y espejo que sobresalía por encima y por detrás de la máscara. Para corregir esta limitación, en el presente trabajo se ha utilizado un micro proyector con las prestaciones necesarias para una correcta proyección y que además sea muy compacto en dimensiones. Después se ha calculado la distancia de proyección mínima y adecuada hasta la máscara. Tras elegir el sistema de proyección, este micro proyector se ha acoplado a un cubículo con la forma del proyector para que quede bien fijado y se ha ubicado dentro del casco que formará la cabeza. Así, un usuario que este observando externamente al robot desde cualquier perspectiva no verá ninguno de los componentes de la proyección y el aspecto visual del robot quedará más natural. Este tema se analiza más profundamente en el capítulo 5.

2.3.4. Avatares SDK

En consideración al diseño únicamente de avatares, la empresa AvatarSDK formada por especialistas en visión por computadora, han conseguido implementar un sistema de generación de avatares extremadamente rápido y preciso. A partir de una única imagen 2D de una persona, este sistema genera en unos pocos segundos, un avatar en 3D extremadamente similar y realista a la persona de la imagen [10]. Las principales características de estos avatares son las siguientes:

- La forma y textura de la cabeza son únicas para cada persona, sintetizadas con un algoritmo de aprendizaje profundo. El algoritmo detecta la silueta de la cabeza y sintetiza una malla en 3D conforme a la silueta, además de detectar todos los colores de los elementos del rostro. En el avatar creado también esta modelado el interior de la boca, y dientes.
- Ejecución de algunas animaciones faciales sobre la malla.
- El algoritmo sintetiza un *mesh* en 3D *midpoly*. Es decir, que la malla poligonal que forma el avatar en 3D está formada por un número medio de polígonos: aproximadamente unos 24000 triángulos. El avatar se ejecuta a una resolución media, y el realismo de la piel en cuanto a reflejos y detalles no es extremo, pero es lo suficientemente realista como para calcar la imagen de la persona.
- Puede funcionar para cualquier PC con Windows 64 bits. También soporta Mac, Linux.

En la figura 2.7, se muestra una prueba en la demo de la página web AvatarSDK, se ha obtenido el avatar virtual a partir de una imagen del autor del presente proyecto.

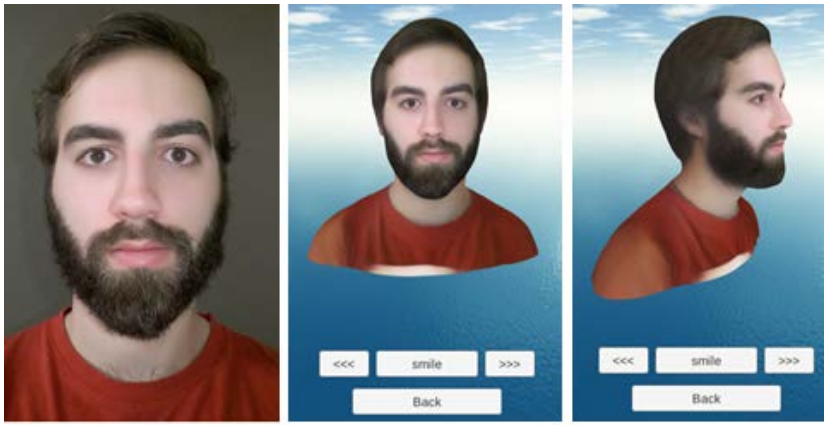


Figura 2.7- Imagen frontal del modelo (izquierda). Avatar 3D generado en la demo (centro y derecha).
Imagen extraída a partir la demo de la web oficial AvatarSDK: https://webdemo.avatarsdk.com/

Después de realizar la prueba sobre la demo del AvatarSDK, se ha llegado a las siguientes conclusiones:

- El avatar visto de frente es casi idéntico a la imagen 2D. El avatar visto de perfil pierde algo de precisión en la silueta, pero a pesar de que no se utilizado ninguna foto de perfil, el algoritmo detecta muy bien la forma y las proporciones de la cabeza.
- Para detectar y distinguir cada una de los elementos de la cara como ojos, boca o la forma de la nariz a partir de una imagen frontal, es posible que dentro del algoritmo implementado se haya utilizado alguna técnica de visión artificial de detección de bordes. De este modo se detectan las siluetas de todos los elementos del rostro, y después se puede dar forma a la malla 3D.
- El avatar puede realizar varias animaciones faciales como sonreír o guiñar un ojo. Aunque las animaciones no queden muy naturales, están bastante bien construidas. Es posible que para las animaciones, los creadores hayan implementado puntos clave en varias zonas en un rostro genérico, como en las comisuras de la boca. Cuando la malla 3D del nuevo modelo se construye, en esos puntos clave se realizan las acciones correspondientes, como por ejemplo estirar o encoger los labios.

Como posible limitación que se ha encontrado en la generación de estos avatares, es que no existe mucha variedad de animaciones que el avatar pueda realizar. Además estas animaciones siempre tienen la misma magnitud de intensidad. Para mejorar esta limitación, en el presente trabajo se ha diseñado un avatar capaz de generar y combinar diversos gestos para formar distintas expresiones con distintos grados de intensidad, tanto en los gestos como en la totalidad de la expresión facial. De este modo el rostro adquiere mayor complejidad y realismo gestual. Este tema se analiza en profundidad en el apartado 3.4 y capítulo 4.

2.4. El rostro humano y las expresiones faciales

El antropomorfismo en los robots se puede definir en términos de apariencia y comportamiento. Respecto a la apariencia, el antropomorfismo indica la similitud de un robot a la figura o aspecto físico de un ser humano. En cuanto al comportamiento, este se puede explicar cómo la identificación de atributos y gestos humanos en el robot. Otro de los conceptos, es el impacto positivo o negativo que puede generar el robot a los usuarios que interactúan con ellos. Esta cuestión es controvertida, y no hay un consenso definitivo [11]. En muchos casos, en el diseño de robots para la interacción humana, se ha empleado el uso de una apariencia menos real y más caricaturesca para justificar el caso del valle inquietante. El valle inquietante, fue postulado por primera vez por Masahiro Mori en 1970. Este término indica que llegar a un grado muy alto de antropomorfismo en el robot genera rechazo al observador [12]. Sin embargo, algunos estudios posteriores [13] señalan que no hay una preferencia universal, si no que el impacto que genera el realismo humano de un robot depende de cada individuo en aspectos psicológicos o emocionales. La edad o el género no tienen apenas efecto en la forma de percibir al avatar/robot. Antes de entrar en el diseño del avatar virtual, se va a estudiar tanto la apariencia del rostro, como el comportamiento definido a través de los gestos y las expresiones faciales del ser humano.

2.4.1. Músculos del rostro humano

Las tres capas más importantes del rostro humano son la piel, los huesos y los músculos. La piel en sus distintas membranas se encarga de las funciones de protección superficial (Epidermis), termorregulación y sensibilidad (Dermis e Hipodermis). En estos últimos tejidos se encuentran las glándulas nerviosas para captar el tacto y las glándulas sudoríparas para oxigenar el cuerpo y segregar el sudor.

Aunque el movimiento de algunos huesos como el maxilar contribuyen a la generación de alguna expresión facial, las funciones de los huesos de la cabeza son la de proteger los órganos sensoriales y cerebro, y producir los movimientos de masticación.

Por último, la estructura muscular es la que se encarga de las expresiones faciales. Los músculos ejercen un movimiento de contracción y relajación, que hace que la piel se arrugue y se pueda percibir visualmente los cambios de aspecto del rostro.

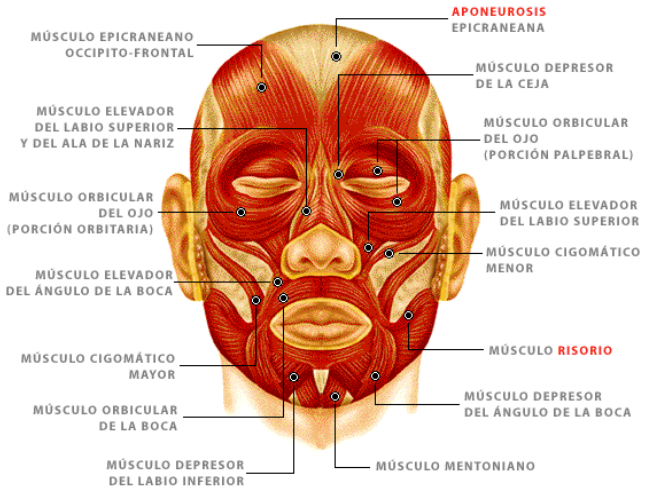


Figura 2.8-Principales músculos faciales.
Imagen extraída de la página web: http://ccnn3eso.blogspot.com/2012/04/principales-musculos-esqueleticos.html

A continuación se citan los principales músculos faciales y el movimiento producido por estos [14]:

- **Musculo elevador del ángulo de la boca:** Levanta y encamina hacia dentro la comisura de los labios. Eleva el ángulo de los labios, empujando la zona interior de las mejillas hacia el ojo.
- **Músculo depresor del ángulo de la boca:** Genera la depresión del borde de los labios al desplazar hacia abaja la comisura labial.
- **Músculo elevador del labio superior:** Dobla hacia afuera el labio superior, levantándolo hacia arriba.
- **Músculo depresor del labio inferior:** Dobla hacia fuera el labio inferior, encaminándolo hacia abajo y hacia fuera.
- **Músculo orbicular de la boca:** Rodea el orificio bucal en forma de elipse. Se encarga del encogimiento y de cierre de los labios, también genera el movimiento para que los labios produzcan la forma de un beso.
- **Músculo risorio:** Retrae la comisura labial. Al contraerse, el diámetro transversal de la boca aumenta, produciendo la sonrisa.
- **Músculo mentioniano**: Levanta la piel del mentón.
- **Músculo Cigomático mayor:** Tracciona supero lateralmente el ángulo de la boca. Elevador de la comisura labial.
- **Músculo Cigomático menor:** Elevador de la parte media del labio superior.
- **Músculo elevador del labio superior y del ala de la nariz:** Eleva el labio superior y ala de la nariz.
- **Músculo orbicular del ojo:** Rodea la zona ocular, en forma de anillo. Es el único musculo capaz de cerrar el ojo.
- **Músculo epicraneano occipito-frontal:** Formado por dos vientres musculares. El vientre occipital y el vientre frontal. Si se contrae el vientre frontal se elevan las cejas y se genera arrugas en la frente. Si se contrae el vientre occipital la piel y aponeurosis se desplaza hacia atrás.

- **Músculo depresor de la ceja:** Al contraerse, mueve las cejas hacia abajo e interior, produciendo arrugas en el comienzo de la frente, en la parte superior de la nariz.

2.4.2. Universalidad de las expresiones faciales

El ámbito de las emociones no es una ciencia exacta. Como consecuencia el estudio de las expresiones faciales asociadas a las emociones ha generado distintos enfoques y conclusiones a lo largo del último siglo y en la actualidad.

El científico Charles Darwin es conocido por ser uno de los científicos más influyentes de la historia. Además de su contribución al estudio de la evolución biológica, en el año 1872 desarrolló varias investigaciones acerca del comportamiento humano. En estas, relacionaba las respuestas faciales con estados emocionales idénticos en todos los seres humanos. Estas ideas están recogidas en la obra "La expresión de las emociones en hombres y animales". Esto le permitió intuir un origen evolutivo de ciertas emociones.

Manteniendo la idea de Charles Darwin, años más tarde Paul Ekman, psicólogo y uno de los pioneros del estudio de las emociones, expuso que las expresiones faciales dependían fundamentalmente de un factor biológico y no tanto de un aspecto cultural. Para ello, en el año 1971 desarrolló una lista básica de expresiones emocionales, y realizó investigaciones con una tribu de Papúa Nueva Guinea [15]. Comprobó que individuos de una tribu pertenecientes a una cultura aislada, conseguían distinguir con fiabilidad las expresiones asociadas a cada emoción de varias imágenes de modelos occidentales. A partir de estos resultados, en el año 1972 Ekman determinó que aunque existen ciertos gestos que pueden variar de unas sociedades a otras, existen seis expresiones faciales básicas, estándar y universales asociadas a las seis emociones principales del ser humano. Estas expresiones son: **alegría, ira, miedo, asco, sorpresa y tristeza.**

Figura 2.9- De izquierda a derecha y de arriba a abajo se muestran las seis expresiones básicas: miedo, sorpresa, alegría, ira, asco y tristeza.
Imagen extraída de la página web:
https://educandoemociones.wordpress.com/2012/09/05/emociones-y-lenguaje/

En la actualidad parte de la comunidad científica sigue considerando válidas las ideas expuestas por Darwin y Ekman acerca de la universalidad de las expresiones derivadas de las seis emociones básicas. Sin embargo, otra parte de la comunidad está rotundamente en contra de esta idea y esta cuestión sigue generando controversia. Los resultados de un interesante estudio realizado por investigadores del Instituto de Neurociencias y Psicología de la Universidad de Glasgow, mostraron que esas seis expresiones consideradas como universales, pueden ser percibidas de forma distinta según diferentes culturas [16] . En dicho estudio, se empleó a 30 voluntarios para actuar de observadores. 15 observadores eran Caucásicos occidentales (CO), y los otros 15 observadores eran de Asia Oriental (AO). El estudio consistía en aplicar a distintos avatares 3D de rostros CO y AO, cuatro unidades de acción de movimiento (UA) aleatorias de entre 41 UA del sistema FACS (apartado 2.4.3). Después, una herramienta generaba la animación temporal del avatar 3D con las cuatro UA aleatorias que se habían aplicado a ese avatar. Por último los 30 observadores tenían que categorizar la animación facial aleatoria con una de las seis expresiones básicas, cuando el movimiento del rostro del avatar correspondiese con la imagen mental del observador de esa expresión facial. Es decir, cuando el observador percibiese una de las seis emociones en el avatar. Cada observador tuvo que categorizar 4800 animaciones faciales de avatares tanto de su misma raza como de la otra raza. En total se computaron 180 modelos mentales de expresiones faciales por cultura (15 observadores x 6 emociones x 2 razas de rostro) y para determinar los resultados se empleó un análisis clúster. Para cada raza se crearon 6 grupos (k=6) correspondientes a las seis emociones básicas, asignando un color a cada grupo (ver figura 2.10). Se emplearon 30 modelos por emoción. El análisis mostró que los modelos mentales producidos por los occidentales formaban seis clúster prácticamente homogéneos, es decir eran capaces de distinguir las seis expresiones básicas. En contraposición, los modelos mentales de los orientales mostraron disparidad, y solapaban algunas emociones como el miedo, sorpresa, ira y asco. Excepto el clúster de alegría y tristeza que eran más homogéneos, los cuatro clúster restantes eran heterogéneos en color.

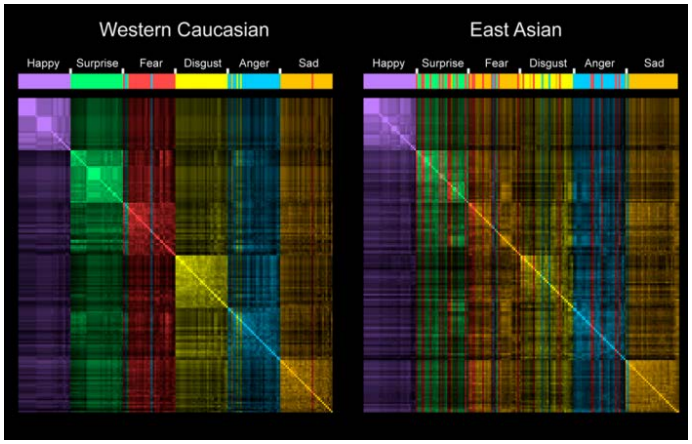


Figura 2.10 – Clasificación clúster de modelos mentales occidentales (izqda) y orientales (drcha). Imagen extraída del proyecto: Facial expressions of emotion are not culturally universal. Autores: Rachael E. Jackab, Oliver G. B. Garrodb, Hui Yub, Roberto Caldarac, and Philippe G. Schynsb

A través de los resultados del estudio, se dedujo que los orientales tienden a fijarse más en la zona superior del rostro, los ojos especialmente, para establecer la distinción de las expresiones y su intensidad. Los occidentales tratan de deducir el grado de las emociones a partir de otros músculos faciales, más en la zona inferior, especialmente los músculos próximos a la boca.

En la figura 2.11 se muestra que es difícil diferenciar las expresiones de asco e ira en el modelo occidental fijándose en los ojos, y más fácil fijándose en la boca. Para el modelo oriental, el asco y la ira se diferencian mejor a través de los ojos y se diferencian muy poco a través de la boca. En los occidentales para ciertas emociones, la zona de la boca es más informativa, y en los orientales lo es la zona de los ojos.

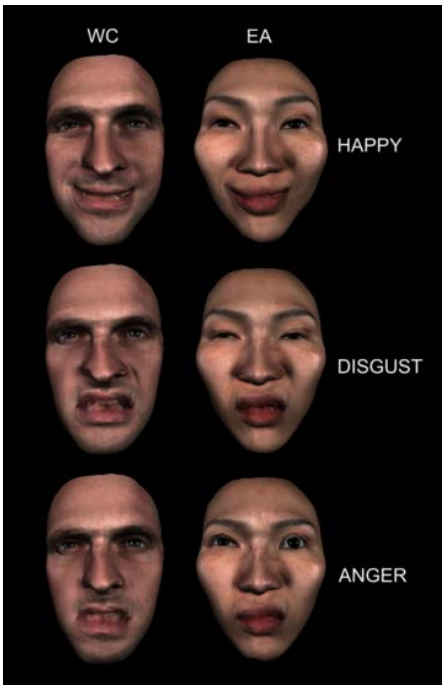


Figura 2.11- Ejemplo de dos modelos (un modelo occidental y otro modelo oriental), para las emociones de alegría, asco e ira.
Imagen extraída del proyecto: Facial expressions of emotion are not culturally universal. Autores: Rachael E. Jackab, Oliver G. B. Garrodb, Hui Yub, Roberto Caldarac, and Philippe G. Schynsb

Como conclusión, comentar que las seis expresiones básicas se pueden considerar universales e innatas ya que el ser humano nació con ellas. Sin embargo, también hay que considerar que los modos de comunicación y la interpretación de las emociones van evolucionando, y que no todas las culturas tienen los mismos rasgos faciales. Los asiáticos tienen los rasgos más marcados en la zona superior de la cara, y por ende durante la comunicación tienden a fijarse más en los ojos para percibir la distinción de las expresiones, ya que es la zona más marcada. En contraste, los rasgos occidentales están más acentuados en la parte media e inferior del rostro, y la forma de percibir las expresiones emocionales es distinta.

2.4.3. Sistema de Codificación Facial

El estudio y análisis del rostro humano para la detección de expresiones y emociones es una labor compleja. Y es que el rostro humano está compuesto por una gran cantidad de músculos, los cuales cada uno pueden generar uno o varios movimientos. Esto añadido a que algunos movimientos pueden ser dependientes de otros, y que cada movimiento se puede realizar con un determinado grado de intensidad, hace del estudio de las expresiones y su distinción una ardua tarea. Un leve cambio en el grado de intensidad de un micro movimiento entre muchos que están actuando a la vez, puede cambiar totalmente la expresión o lo que quiere transmitir la persona emisora.

Otra de las complejidades reside en que los seres humanos no realizan las expresiones faciales idénticamente. Todos los seres humanos tienen la capacidad de mover los mismos musculares faciales para realizar multitud de expresiones, pero aparte de los movimientos genéricos, cada individuo puede tener sus micro-movimientos característicos. De la misma manera que el aspecto físico, la expresión facial es lo que hace que las personas se vean únicas y diferentes unas de otras.

A lo largo del siglo XIX, anatomistas y científicos estudiaron esta temática, y la forma de establecer un método o reglamento universal para detectar y clasificar las expresiones para cualquier individuo. En el año 1978, el científico W.V. Fiesen y el doctor P. Ekman, propusieron el sistema de Codificación Facial o FACS. Este sistema desarrolla un método universal y estándar para la codificación de los gestos faciales del rostro humano, y así clasificar sistemáticamente las expresiones faciales de las emociones o estados de ánimo. A día de hoy y después de algunas actualizaciones, el sistema de codificación facial sigue siendo un método de reconocimiento bastante robusto, y tiene utilidad en varios campos como la psicología o la animación. FACS se ha implantado como un sistema automático que detecta las características del rostro para posteriormente reproducir cada movimiento facial.

2.4.3.1. Unidades de Acción

El FACS indica que el rostro humano puede dividirse en un conjunto de unidades de movimiento aisladas, que en combinación con otras unidades, generan una determinada expresión facial. Estas unidades de movimiento se denominan unidades de acción, UA. La unidad de acción, formada por un solo movimiento, se considera como la unidad menor de movimiento perceptible, perteneciente a un determinado grupo muscular del rostro.

La relación entre las unidades de acción y los músculos faciales no es 1:1 [17]. Esto significa que no todas las unidades de acción se corresponden exactamente con el movimiento de un músculo en concreto de la cara. Esto puede ocurrir porque la acción de ciertos músculos actuando individualmente sea apenas perceptible y se requiera más de un músculo para percibir un movimiento, por lo que la unidad de acción pueda englobar más de un músculo. O puede ocurrir lo contrario, que un mismo músculo se divida en varias UA, ya que el músculo permita más de un movimiento. Un ejemplo de este argumento se da en las unidades de acción AU1 y AU2. Ambas unidades de acción actúan en el mismo músculo, pero cada una se encarga de un movimiento. La AU1 representa la contracción de la porción medial del frontal, para elevar la parte interior de las cejas. La AU2 representa la contracción de la porción lateral del frontal, para elevar la parte exterior de las cejas.

El FACS, es un método adecuado y eficaz para interpretar las expresiones faciales, pero no es un sistema cuyo objetivo sea el de proporcionar una información biomecánica exacta de los músculos del rostro.

Para diferenciar y etiquetar cada una de las UA bajo una pauta estándar y universal, el FACS ofrece una lista de todas las unidades de acción. A cada unidad de acción se le asigna un número, aunque este número no sigue un orden dado de grupos de músculos. En las tablas 1, 2 y 3 se detallan las UA con la etiqueta, la acción correspondiente, y los músculos implicados en dicha acción.

Número de acción	***Nombre de la acción***	***Músculos / articulación que intervienen***
0	*Rostro Neutral*	
1	*Levantamiento interior de ceja*	*Frontal (par medial)*
2	*Levantamiento exterior de ceja*	*Frontal (par lateral)*
4	*Bajar cejas*	*Músculo depresor de la ceja*
5	*Levantamiento del párpado superior*	*Elevador del párpado superior*
6	*Levantamiento de mejillas*	*Orbicular del ojo (par orbital)*
7	*Apretar parpado(s)*	*Orbicular del ojo (parpebral)*
9	*Arrugar la nariz*	*Elevador del labio superior*
10	*Levantamiento del labio superior*	*Elevador del labio superior, alar de la nariz*
11	*Profundidad nasolabial*	*Elevador del labio superior*
12	*Tiramiento labial esquinal*	*Cigomático mayor*
13	*Tiramiento labial frontal*	*Triangular*
14	*Hoyuelo facial*	*Buccinador*
15	*Depresión labial esquinal*	*Triangular*
16	*Depresión labial frontal*	*Depresor del labio inferior*
17	*Levantamiento de barbilla*	*Borla*
18	*Contraer labios adelantado*	*Incisivo del labio superior,□incisivo del labio inferior*
20	*Apretar los labios*	*Risorio*
22	*Embudo labial*	*Orbicular de la boca*
23	*Morder labios*	*Orbicular de la boca*
24	*Presión Labial*	*Orbicular de la boca*
25	*Deslizamiento labial*	*Orbicular de la boca*
26	*Caída de la mandíbula*	*Articulación temporomandibular*
27	*Apretamiento bucal*	*Pterigoideo*
28	*Lamido labial*	*Orbicular de la boca*
29	*Tracción de la mandíbula*	*Temporal*
41	*Bajada de párpados*	*Orbicular del ojo*
42	*Contracción retinal*	*Orbicular del ojo*
43	*Ojos cerrados*	*Orbicular del ojo*
44	*Recolector retinal*	*Orbicular del ojo*
45	*Parpadeo*	*Orbicular del ojo*
46	*Guiño*	*Orbicular del ojo*

Tabla 1 – Principales unidades de acción de la cara

Número de acción	***Nombre de acción***	***Músculos***
51	*Girar cabeza a la izquierda*	*Músculos del cuello*
52	*Girar cabeza a la derecha*	*Músculos del cuello*
53	*Alzar la cabeza*	*Músculos del cuello*
54	*Bajar la cabeza*	*Músculos del cuello*
55	*Inclinación de la cabeza hacia la derecha*	*Músculos del cuello*
M55	*Inclinación de la cabeza a la izquierda*	*Músculos del cuello*
56	*Leve inclinación*	*Músculos del cuello*
M56	*Inclinación derivada en el cuello*	*Músculos del cuello*
57	*Head Forward*	*Músculos del cuello*
M57	*Empujar cabeza hacia adelante*	*Músculos del cuello*
58	*Empujar hacia atrás*	*Músculos del cuello*
M59	*Agitar cabeza hacia arriba y abajo*	*Músculos del cuello*

Tabla 2 –Unidades de acción del cuello

Número de acción	***Nombre de acción***	***Músculos***
61	*Mover ojos hacia la izquierda*	*Músculos extraoculares*
62	*Mover ojos a la derecha*	*Músculos extraoculares*
63	*Ojos hacia arriba*	*Músculos extraoculares*
64	*Ojos hacia abajo*	*Músculos extraoculares*

Tabla 3 – Unidades de acción de la vista

2.4.3.1.1. Intensidad de las Unidades de Acción

Todas las unidades de acción constan de varios niveles de intensidad. Estos niveles de intensidad sirven para indicar el grado o magnitud de movimiento de la acción o del gesto facial. Los niveles de intensidad abarcan desde el movimiento más leve de la acción hasta el movimiento máximo de la unidad de acción. En el FACS se plantearon cinco niveles, expresados mediante letras desde A-E:

- A: Trazo
- B: Leve
- C: Pronunciado
- D: Extremo
- E: Máximo

Ejemplo: AU15E indica que la unidad de acción es la depresión labial esquinal, y el movimiento de la acción ha llegado al punto máximo.

La integración de los valores de intensidad es importante, ya que dos expresiones formadas por las mismas unidades de acción no tienen por qué representar o significar lo mismo. Si se desea mostrar muy fuerte una emoción, los gestos que forman la expresión han de tener un nivel de intensidad máximo. Pero puede darse el caso de querer expresar la misma emoción con menos pasión o de forma más contenida, por tanto ha de ser más leve el nivel de intensidad de las UA que forman la expresión de dicha emoción.

2.4.3.1.2. Evolución temporal de las Unidades de Acción

Cada una de las unidades de acción sigue una evolución temporal desde el reposo hasta un cierto nivel de intensidad. La evolución de cada acción pueda estar definida a partir de cualquier función temporal ya sea lineal con mayor o menor pendiente, o curva. Si se desea conseguir que un gesto o expresión facial se realice de forma natural, la función de las unidades de acción ha de ser una curva similar a la representada en la figura 2.12. El paso de una intensidad baja a una intensidad máxima no se realiza de forma brusca, y no se emplea el mismo intervalo de tiempo para todos los niveles. Normalmente el nivel C y D son los niveles que abarcan el mayor tiempo.

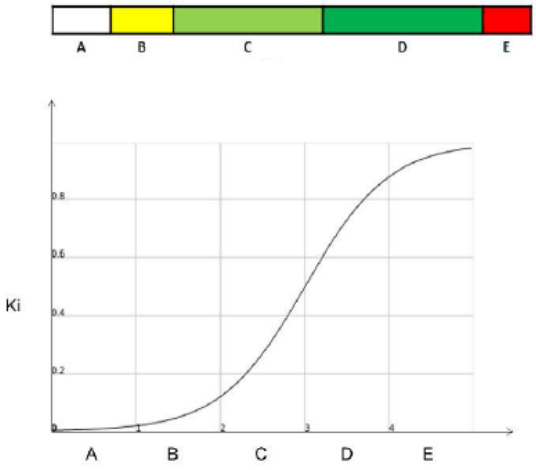


Figura 2.12-Curva que representa la evolución temporal ideal de una UA.
Imagen extraída del proyecto [25]. *Se empleó en dicho proyecto una curva similar para definir el movimiento de los servomotores asociados cada uno a una UA.*

.

2.4.3.2. Construcción de las expresiones faciales estándar mediante FACS

La combinación de una o varias unidades de acción genera una expresión facial. Teniendo en cuenta el número de UA definidas en el FACS existen multitud de expresiones que se pueden formar. Por esa razón el FACS es un sistema bastante eficaz para la clasificación o la detección de expresiones. En este proyecto se van a emplear principalmente las expresiones definidas en el EMFACS *(Emotional Facial Action Coding System):*

Emoción	Unidades de acción
Felicidad	6+12
Tristeza	1+4+15
Sorpresa	1+2+5B+26
Miedo	1+2+4+5+7+20+26
Coraje	4+5+7+23
Disgusto	9+15+16
Desprecio	R12A+R14A

Tabla 4 – Expresiones emocionales a partir de unidades de acción
Imagen extraída de [26]

2.4.3.3. Alcance del FACS

La implementación de las UA junto a sus niveles de intensidad, permite dar solución al problema que se trataba al comienzo del apartado 2.4.3. El modo de determinar expresiones análogas pero que difieren en pequeños micro movimientos que cambian el sentido de la expresión.

Un ejemplo son los dos tipos de sonrisa: La sonrisa de Duchenne y la sonrisa Non-Duchenne [18] [19]. La sonrisa Duchenne involucra al músculo cigomático mayor y menor para elevar las comisuras de los labios, y el orbicular del ojo, que al contraerse, hace que las mejillas se elevan y se arrugue la zona del parpado inferior. El médico Guillaume Duchenne descubrió que el ser humano realiza esta expresión de forma espontánea y sincera, y que tiene un origen neurológico derivado de un impulso producido en los núcleos basales. Se considera espontánea, ya que es una tarea difícil el contraer de forma voluntaria algunos de los músculos implicados. Unidades de acción implicadas: AU12+AU25+AU6. Sin embargo la sonrisa non-Duchenne solo involucra al músculo cigomático mayor para elevar las comisuras de los labios. A diferencia de la anterior, esta sonrisa se puede realizar voluntariamente y tiene su origen en la corteza motora. Unidades de acción implicadas: AU12+AU25.

Con la codificación facial se puede entonces diferenciar dos expresiones que se perciben como semejantes, pero su significado es diferente. Esto es de gran utilidad para aplicaciones de sistemas de detección facial, con el fin de determinar la sinceridad o la falsedad de las intenciones de una persona mediante las expresiones del rostro.

Figura 2.13- Representación de los dos tipos de sonrisa en dos modelos distintos, y las unidades de acción asociada a estas.
Imagen extraída del artículo: https://www.frontiersin.org/articles/10.3389/fpsyg.2017.01374/full#B49

Capítulo 3 – Desarrollo del Avatar

3.1. Introducción

En el siguiente capítulo se va a exponer el desarrollo del avatar virtual, incluyendo el modelado, el texturizado y las animaciones faciales. Antes de entrar en el desarrollo, se ha dedicado un apartado a analizar y explicar brevemente las herramientas más importantes del entorno empleado para el modelado.

3.2. Entorno de modelado Blender

Blender es un programa orientado al diseño, modelado, texturizado, animación y renderizado de gráficos tridimensionales. A diferencia de otros programas de animación, Blender es gratuito y es de distribución libre, se puede utilizar y modificar libremente. Lleva implementado un intérprete de Python, que se carga al iniciar Blender y permanece activo mientras Blender este ejecutándose. Este intérprete permite ejecutar scripts en Python para editar la interfaz de usuario o cargar nuevas herramientas de trabajo en forma de *addons* o extensiones. Es compatible con todas las versiones de Windows, Mac OS X, GNU/Linux entre otros sistemas operativos.

Este es el entorno que se va a emplear para el diseño del avatar y la máscara. La interfaz es algo compleja si previamente no se está familiarizado con el programa ya que presenta muchos editores, características y comandos de uso que lo hacen bastante completo en el ámbito del modelado y diseño 3D. En este apartado se explicarán las zonas de trabajo más importantes del programa, con especial interés en las zonas que se han empleado para desarrollar el avatar. También se nombrarán algunos conceptos importantes del programa, y en los anexos se han redactado unas tablas con los comandos útiles para trabajar en Blender. La información consultada en este apartado para el desarrollo del avatar procede principalmente del manual de Blender [20].

3.2.1. Conceptos clave en Blender

En apartados posteriores se van a citar algunos conceptos importantes de Blender. Para evitar explicaciones reiterativas se ha considerado preferible reunir en este apartado, por orden alfabético, algunos conceptos básicos que se van a definir brevemente:

Acción: La acción de un objeto es el cambio en las propiedades de movimiento del mismo entre dos o más fotogramas distintos. Para crear y editar acciones se emplea el editor *Dope Sheet*.

Animación: Conjunto de acciones realizadas en una escala espacial, cuya unidad principal es el fotograma, en un tiempo determinado medido en segundos.

Escena: Espacio donde están ubicados todos los objetos creados: figuras geométricas, lámparas, cámaras etc. La escena se puede visualizar en el editor *Vista 3D*.

Extrusión: Operación de modelado que permite generar una nueva forma poligonal a partir de la forma poligonal seleccionada. Los vértices seleccionados para extruir se duplican, y la nueva forma poligonal se mantiene conectada a la original mediante aristas o caras.

Driver o controlador: Herramienta para controlar los valores de las propiedades de un objeto, a partir de las propiedades de otro objeto o a partir de una función o expresión matemática. Cualquier propiedad del objeto puede ser controlada. El driver establece una interpolación entre los valores inicial y final de ambas propiedades. De esta forma la propiedad de un objeto A puede estar controlado en todo momento por otra propiedad distinta de un objeto B.

Forma clave: Deformación en la malla de la forma base de un objeto (apartado 3.4.1).

Fotograma: Imagen estática que al estar en secuencia con otros fotogramas a una determinada frecuencia de fotogramas por segundo, genera la sensación de movimiento. En Blender puede editarse cada fotograma para generar distintas acciones.

Huesos: Elementos que se pueden implementar en la escena, y su principal utilidad es controlar el movimiento de las partes de un objeto (apartado 3.4.2).

Keyframe: Herramienta que permite guardar las propiedades de movimiento de un objeto de la escena y asignarlas a un fotograma de la animación, llamado fotograma clave.

Mapa UV: Mapa 2D de cualquier objeto tridimensional. Este mapa va a ser de utilidad para texturizar la superficie o malla del objeto.

Mesh: Figuras geométricas básicas (circulo, cuadrado, esfera, toroide…) que Blender proporciona para que el usuario pueda manipularlas y editarlas y crear así objetos o figuras más complejas. El modelado de malla en Blender comienza con una forma primitiva de malla, esta forma primitiva es el *mesh*.

Unidades de acción: Movimientos aislados de un objeto, que se pueden combinar entre sí para generar movimientos más complejos.

3.2.2.Zonas de trabajo

Las zonas de trabajo se considera que son los distintos editores, ya que cada uno sirve para un propósito específico. En Blender se pueden abrir varios editores distintos a la vez, y el programa permite configurar la interfaz como el usuario desee, lo que proporciona flexibilidad y facilidad a la hora de trabajar. *Type editor* menú es el menú general de Blender. Muestra todo los editores con los que se puede trabajar.

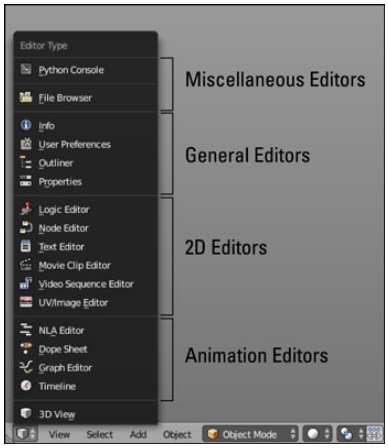


Figura 3.1- Menú de editores de Blender

3.2.2.1. Editores Generales

User preferences: Ofrece acceso a todas las características de programa que el usuario puede modificar. Se puede configurar el sistema gráfico y de audio, características de la interfaz, y los *addons*. Los *addons* son extensiones que sirven como herramientas de todo tipo en Blender: herramientas de modelado, texturizado, animación etc. Estos están implementados en *scripts* de Python, muchos de ellos creados por usuarios de la comunidad. Es por eso que se pueden cargar nuevos *addons* desde fuera de Blender.

Todos ellos están organizados en una lista, y el usuario puede añadirlos o quitarlos. Si los *addons* están activos estos aparecerán como ventanas o botones dentro del editor correspondiente.

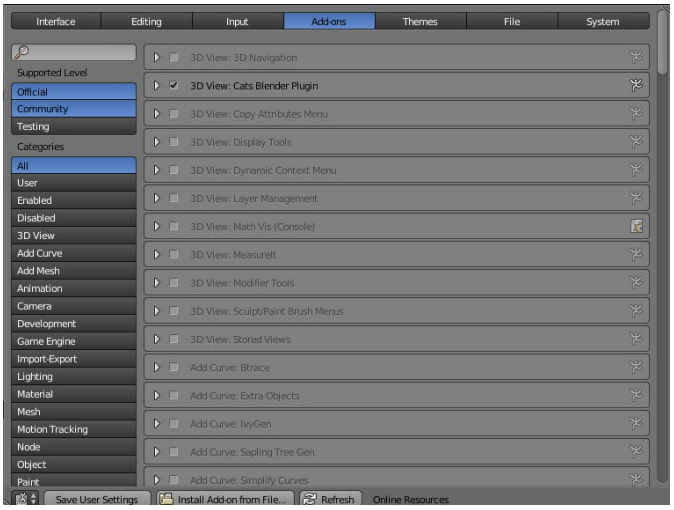


Figura 3.2- Preferencias de Blender

Outliner: Editor que muestra en formato ramificado todo lo que el usuario ha introducido en la escena: objetos, hueso, cámaras, lámparas etc. Dentro de cada objeto, se muestra las propiedades que el objeto tiene implementado: desde la malla primigenia, el material o textura, modificadores y muchas más propiedades. En el apartado *data-blocks*, se muestran en una lista todos los datos guardados del modelo con el que se está trabajando. Desde la posición de cada uno de los vértices de todos los objetos del modelo, hasta un registro con todos los materiales empleados, texturas, animaciones y muchos otros datos. Es decir todo lo que el usuario ha editado hasta el momento.

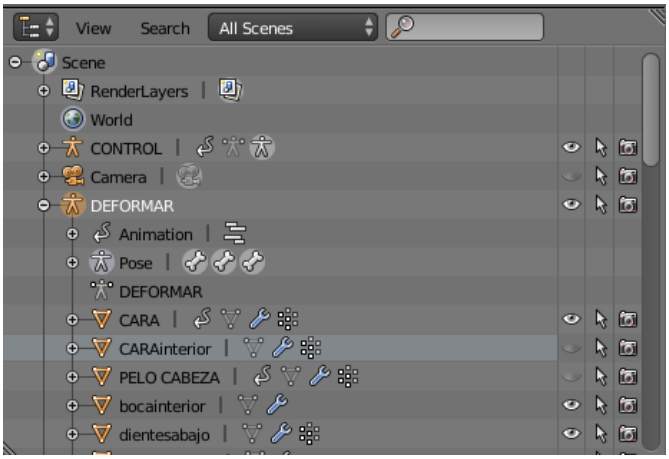


Figura 3.3- Editor Outliner

Properties: Editor que permite cambiar y modificar las propiedades del objeto seleccionado, además de añadir nuevas propiedades. También muestra los datos relacionados con el propio objeto seleccionado. Este editor está compuesto por varios apartados:

- *Render:* Se muestran las características del renderizado del objeto: Incluye la ejecución de la animación renderizada, la inclusión de archivos de audio, y se puede editar desde el tamaño de imagen del renderizado, la resolución de la imagen, ratio de fotogramas por segundo, el fotograma inicial y final de la animación, y funciones de iluminación durante la renderización.
- *Scene y World*: Muestra las características del escenario donde se encuentra el modelo. Se puede editar el *background* de la escena, editar las unidades métricas empleadas, incluir efectos como la gravedad además de otras características.
- *Object:* Incluye todos los datos físicos del objeto: la ubicación, movimientos, la posición, la rotación y el escalado del objeto en las coordenadas cartesianas. También se incluye el modo de rotación del objeto, el bloqueo de transformación de ejes para la traslación rotación o escalado. La capa en la que se encuentra el objeto. La relación que este objeto tiene con otros elementos de la escena. Y características de visualización en el *display*: si las coordenadas del objeto son visibles o si el modelo en modo objeto se muestra en formato de malla o sin la malla.
- *Restricciones:* Sub-apartado donde se pueden añadir restricciones de movimiento al objeto. Estas restricciones incluyen: restricción de movimientos de traslación, rotación o escala entre unos valores limites, restricciones de *tracking* o seguimiento a otros objetos, y restricciones en cuanto a la relación con otros objetos (relación padre-hijo entre objetos). Los huesos también constan de un apartado de restricciones.
- *Modificadores*: Permite añadir y editar modificadores al objeto. Los modificadores son operadores de modelado que se pueden añadir o quitar al objeto sin afectar al modelo original, aunque también se puede configurar para que el efecto del modificador sea permanente. Los modificadores incluyen efectos procedurales de edición y deformación de la malla del objeto. Un ejemplo de modificador bastante empleado es: *"Mirror"*. Este modificador, a partir del eje de simetría del objeto, permite generar automáticamente nuevos vértices en la posición simétrica del objeto respecto al eje. Herramienta útil si se quiere crear objetos simétricos, de este modo solo hay que modelar un lado del objeto, el otro lado se genera idénticamente. Otro ejemplo es el modificador "*Subsurf*" que aumenta la subdivisión de los polígonos de la figura, generando más vértices. Este efecto puede aumentar la precisión a la hora de modelar figuras complejas, pero también aumenta el coste computacional a la hora de renderizar el objeto.
- *Datos:* En este sub-apartado se incluyen todos los datos de edición del objeto. Se incluye el *mesh* original a partir del cual se ha creado el objeto, una tabla con grupos

de vértices y una tabla con todas las formas clave que se han implementado en el objeto.

- *Texturas:* Sub-apartado para crear texturas en un objeto. Las texturas se implementan en la superficie del objeto 3D para modificar el patrón, y el aspecto visual del objeto.
- *Materiales:* Sub-apartado para editar y añadir materiales al objeto. Los materiales están formados por texturas más *shaders* y otras propiedades. El material de un objeto es la parte que muestra las propiedades ópticas del objeto frente a fuentes de luz externas. Los materiales se implementan en la superficie del objeto 3D.
- *Partículas:* Sub-apartado para implementar partículas en el objeto. Los tipos de partículas son *emitter* y *hair*. *Emitter* permite añadir la emisión o producción de partículas a lo largo de un periodo de tiempo. *Hair* permite añadir pelo, en una determinada región y añadir la cantidad deseada.
- *Físicas*: Sub-apartado donde se permite editar fuerzas que actúan sobre el objeto como colisiones, fuerzas de campo, humo o efectos de fluidos.

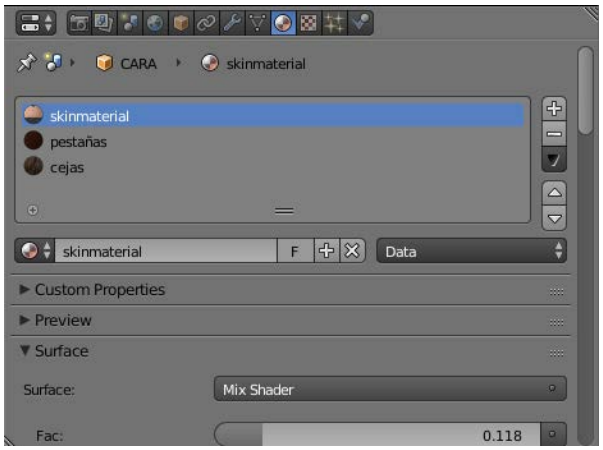


Figura 3.4 – Editor de propiedades

3.2.2.2. Editor 3D

3D View editor: Editor principal para el modelado de un objeto. En este editor se visualiza la escena que se está diseñando en 3D, con todos los objetos, cámaras etc. En este editor se va modificando el aspecto del objeto tridimensional, y se perciben visualmente los cambios.

3.2.2.3. Editores 2D

Text editor: El editor de texto permite crear *scripts* en Python, y después ejecutarlos en el propio modelo de Blender. La característica de Blender es que todo lo que se puede implementar mediante la interfaz 3D, también se puede implementar mediante *scripts*, incluidos menús y paneles de la interfaz.

Logic editor: Editor que se encarga de configurar la lógica o control de los objetos del modelo para el motor de juego o animación. Está formado por bloques que se interconectan en tres columnas. La primera columna es la columna de sensores. Los sensores son los elementos que se utilizan para activar el control del objeto. Los sensores pueden ser eventos incluidos dentro de la propia animación como pueden ser colisiones, o pueden venir de un hardware externo como la configuración de un conjunto de teclas de un *joystick* o teclado como método de activación. En la segunda columna se sitúan los controladores. Los controladores establecen el sistema de comunicación entre los sensores y actuadores a los que están conectados. Primero guardan los datos que reciben de los sensores. Después el usuario puede configurar la lógica en la que operan los controladores. La lógica puede ser mediante un operador lógico (AND, OR…), mediante una expresión o mediante un script de Python. Después de realizar las operaciones lógicas que se han especificado, el controlador envía la señal a los actuadores a los que esta conectados para accionarlos.

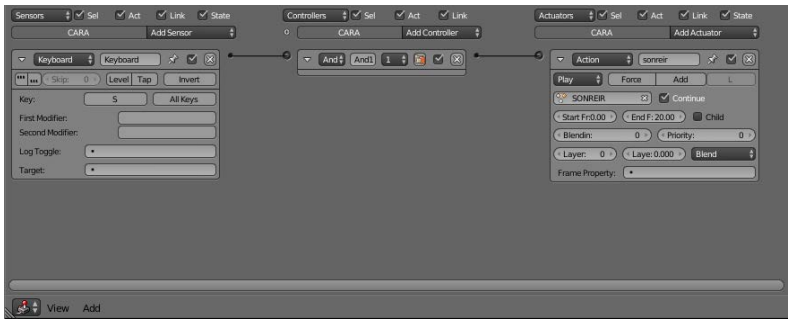


Figura 3.5 – Editor lógico

Node editor: La red de nodos es generalmente utilizada para la creación de materiales asociados a un objeto. El editor de nodos está formado por tres tipos de nodos: nodos de entrada, nodos de salida y nodos intermedios.

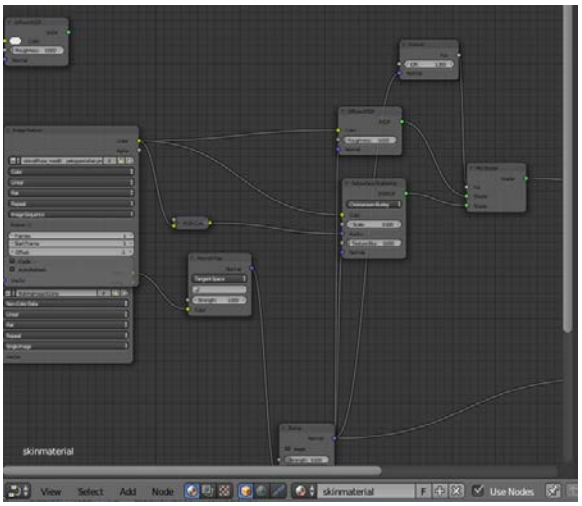


Figura 3.6 - Editor de nodos

Video sequence editor: Su función principal es editar videos. Consta de una herramienta llamada VSE o editor de secuencia de video. Esta herramienta está compuesta por un completo sistema de edición de video que puede combinar muchos canales de video y agregar efectos sobre estos. Este editor no se ha utilizado para este trabajo.

Movie clip editor: Este editor sirve para la operación de rastreo o enmascaramiento en la realización de películas. La operación de rastreo se utiliza para el seguimiento de los movimientos de objetos o cámara. Con el enmascaramiento se puede extraer un objeto particular del metraje. Este editor no se ha utilizado para este trabajo.

Uv/Image editor: Editor de imágenes y de mapas UV. La primera de las operaciones a seguir a la hora de implementar una textura, es desenvolver el mapa 3D de la malla del objeto en un mapa 2D para poder asociar a cada coordenada o forma poligonal del objeto un fragmento de textura. La textura no se puede implementar directamente en la superficie 3D del objeto. Para realizar este proceso se emplea este editor. En el editor se muestra el mapa 2D del objeto y /o la imagen que se emplee como textura.

3.2.2.4. Editores de animación

Estos editores son los encargados de programar la animación de cualquier objeto que aparezca en escena. Cada editor tiene un propósito específico dentro de la animación.

Graph editor: En este editor es donde se implementa el control de las unidades de acción o formas clave. Para realizar este control y asignar los controladores se emplean las curvas F. Las curvas F son gráficas que relaciona por un lado la propiedad del objeto a controlar y la propiedad del objeto que controla. Mediante interpolación se establece esta relación de control.

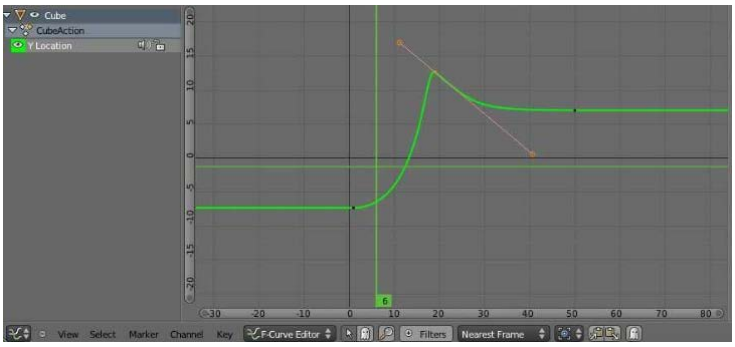


Figura 3.7- Editor de Gráficas

Dope Sheet: Editor que se encarga de crear las acciones. Permite asignar en cualquier fotograma un *keyframe* del objeto, y que durante la animación, se muestre el movimiento producido por el objeto entre los distintos fotogramas. Proporciona una vista panorámica de todos los fotogramas de la escena. En este editor se pueden juntar y modificar varias unidades de acción a la vez para crear un conjunto de acciones concurrentes.

Figura 3.8 - Editor de acciones en Dope Sheet

NLA Editor: Este editor proporciona una vista general de todas las acciones que forman la animación completa. En este editor se pueden manipular y reutilizar acciones ya creadas sin tener que manejar cada fotograma individual. Para editar cada una de las

acciones se utiliza *dope sheet*. Para combinar o duplicar varias acciones distintas a lo largo del tiempo se emplea este editor.

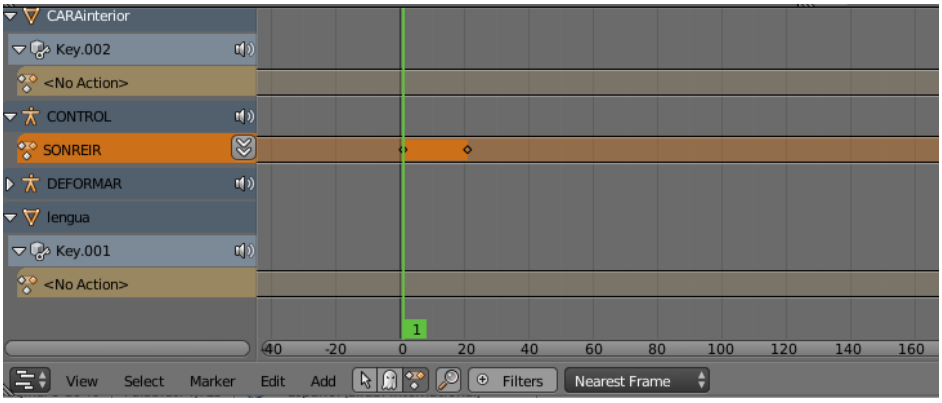


Figura 3.9 – Editor NLA

Timeline: Muestra en una línea temporal todos los fotogramas de la animación. En este apartado se puede establecer el fotograma inicial y final de la animación, y reproducir hacia delante o hacia atrás a mayor o menor velocidad la animación. También se pueden incluir marcadores, para denotar eventos o fotogramas importantes dentro de la animación.

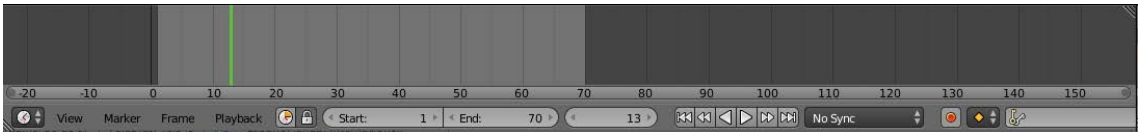


Figura 3.10 - Timeline

3.2.2.5. Editores miscelánea

El editor *filebrowser* se utiliza únicamente para cargar archivos existentes de Blender. El editor *Python Console*, es un editor formado por una consola que permite ejecutar de forma rápida comandos con acceso completo a la API de Python e historial de comandos.

3.3. Diseño del avatar

3.3.1. Modelado del avatar

Una vez analizadas las herramientas y los editores principales que componen el entorno de trabajo, en este apartado se va a proceder a realizar el diseño y modelado del avatar virtual. En la realización de este modelo, se han utilizado dos imágenes del rostro del autor como base. Aunque el avatar final no va a ser igual que el rostro de la foto, el uso de estas imágenes facilita la realización del avatar para mantener las proporciones del rostro humano. Para ello se han empleado dos fotos: una correspondiente al plano frontal y otra en el plano de perfil:

Figura 3.11 – Fotos para modelar el avatar

Debido a que varios puntos de la cara en las dos imágenes no están proporcionados en una primera instancia, mediante el programa Photoshop se ha empleado la función de escalado de imágenes, hasta conseguir que puntos esenciales del rostro se encuentren a la misma altura:

Figura 3.12 – Ajuste de la proporción mediante Photoshop

Con las dos imagenes ya proporcionadas, es importante configurar la posicion de las dos imágenes, de forma que el rostro en las dos imágenes este a la misma altura espacial dentro de la ventana 3D. Esto es con el fin de que cuando se modifique el objeto desde una perspectiva, se mantenga ese objeto en la misma posicion espacial si despues se visualiza desde otra perspectiva.

Figura 3.13- Colocación de las imágenes en Blender a la misma altura espacial

Una vez configuradas las imágenes de plantilla, ya se comienza realmente con el diseño del avatar. Al tener la referencia de dos imágenes, se va a trabajar en dos ventanas 3D. Una de estas ventanas se emplea para visualizar el diseño del avatar desde una vista frontal, que es donde se situa la imagen frontal como plantilla. La otra ventana muestra la vista de perfil izquierdo, que es donde se carga la imagen del perfil izquierdo como plantilla.

Para mantener una organizacion en el modelado, se ha seguido un orden descendente comenzado con el diseño de la esfera de los ojos y el contorno de los parpados. A continuacion se ha diseñado la nariz y los pómulos; despues la boca y el mentón; y por último se ha procedido a completar la frente, modelar las orejas, el interior de la boca y cerrar el cráneo.

3.3.1.1. Párpados y ojos

Como se ha descrito en el apartado de conceptos básicos de Blender, se puede diseñar cualquier forma posible tanto en 2D como en 3D a partir de unas formas y figuras básicas que son los *mesh*. También cabe añadir que todo lo que se modele está formado por vértices, aristas y caras y se puede trabajar de la forma que el diseñador desee. Hay ocasiones en las que es más práctico modelar a partir de vértices, otras veces a partir de aristas y en otras situaciones es mejor emplear las caras. Pero cualquier objeto se puede modelar empleando cualquiera de las formas básicas de la geometría.

Para el diseño de los párpados se comienza añadiendo en modo objeto un *mesh*, en este caso va a ser un círculo. En modo edición, este círculo se va a editar hasta conseguir la forma del contorno de los ojos o lo que será la periferia de la cavidad ocular.

Para generar los párpados hay que extruir varias veces el contorno anterior para generar una malla con forma de antifaz, que será la región donde se sitúen los párpados. Esta nueva forma está conectada a la original mediante aristas. En la figura 3.14 se muestra el proceso de extrusión del contorno de los ojos. Esta extrusión se ha realizado junto con un escalado mayor que uno, por lo que se generan nuevos polígonos hacia el exterior de la geometría original. Así se ha generado la región exterior a la cavidad del ojo.

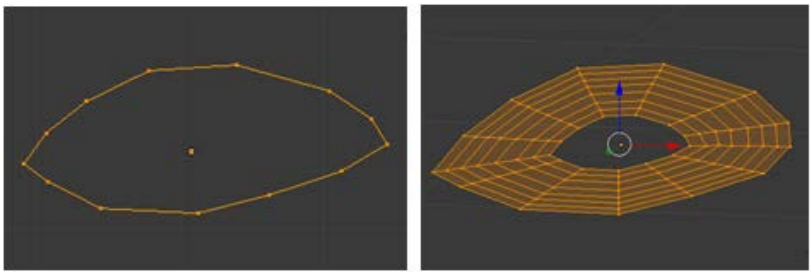


Figura 3.14 – Proceso de extrusión. Izquierda: Forma poligonal original. Derecha: Forma poligonal extruida siete veces.

Siguiendo con el proceso de extrusión y unión de vértices se acaba generando la forma de un antifaz, que será la zona de los párpados y la parte superior del tabique nasal.

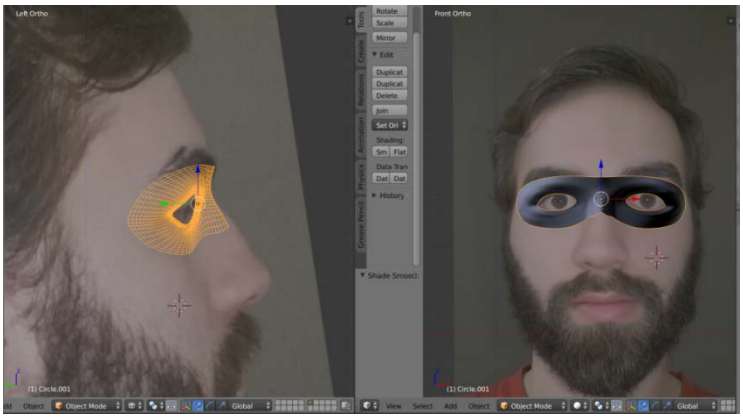


Figura 3.15 – Malla creada en la zona de los párpados

El modelado de los ojos, al igual que más adelante el interior de la boca se ha diseñado separado de la configuración del rostro. Es decir, se ha creado otro objeto a partir de otro *mesh*. Esto es para facilitar el diseño y posterior texturizado de todos los elementos que componen la cara del modelo. Para realizar el ojo se parte de una esfera. En modo objeto se selecciona el *mesh* esfera. Después en modo edición, se comienza a modelar el ojo. La superficie de la esfera será la esclerótica del ojo, la membrana blanca que constituye la parte externa del globo ocular. Para crear el iris se han realizado dos extrusiones donde las formas geométricas estén muy próximas entre sí. De esta forma se diferencia la zona del iris y la esclerótica. Por ultimo para delimitar la zona de la pupila con el iris se realiza otro par de extrusiones hacia dentro, de forma que se genera una pequeña cavidad o hueco. Esa pequeña cavidad o hueco será la zona de la pupila. En la figura 3.16 se muestra el diseño. Por último se ha seleccionado todo el objeto y se ha duplicado para crear el mismo objeto idéntico, que será el otro ojo.

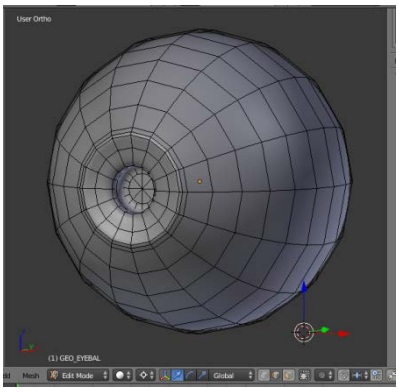


Figura 3.16 – Modelado del ojo

3.3.1.2. Nariz y Labios

La nariz se va a modelar a partir del *mesh* con forma de antifaz que se creado anteriormente. Aplicando la operación de extrusion hacia abajo, se sigue generando la malla. Prestando atencion a la imágenes plantilla se va configurando la forma de la nariz, moviendo los vertices hasta ubicarlos en una posicion adecuada. Una vez diseñada la forma de la nariz se selecciona la zona circular donde van a estar ubicados los dos cavidades nasales y se realiza un proceso de extrusión hacia dentro para formar los orificios externos de la nariz.

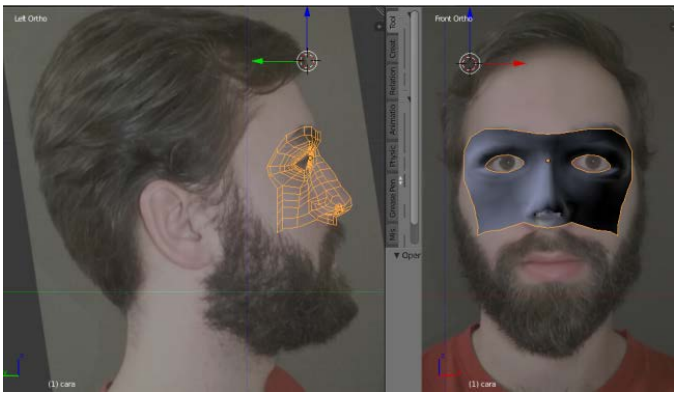


Figura 3.17 – Malla de la zona media-superior del rostro: párpados y nariz

Para crear la forma de los labios se ha añadido un mesh círculo. Este círculo se ha escalado en el eje X hasta componer la forma de una elipse muy excéntrica, que casi sea una línea, pero sin que se toquen los puntos opuestos de la elipse. Esta forma elipsoide serán los límites del labio superior e inferior. Tras esta operación se ha extruido la elipse hacia afuera (extrusión mas un escalado mayor que 1). Se ha realizado esta operación tres veces hasta generar las distintas zonas o capas labiales. Para terminar se han retocado los vértices para modelar el surco del labio superior, y las comisuras derecha e izquierda de los labios .

Una vez diseñada la zona labial se realizan mas extrusiones con escalado mayor que uno, para generar malla y formar el mentón, y a su vez aproximar la parte de los labios y la nariz. Para unificar dos vértices de distinta malla, se seleccionan los vértices y se realiza la unificación (tecla F). Si se tienen tres o mas vértices y se deseea que estos formen una cara, se seleccionan los vértices deseados y se unifican de forma análoga. Realizando esta operación se ha unificado en una sola malla, la zona superior que constituyen los párpados y la nariz, y la zona inferior que constituye la parte de los labios y el mentón.

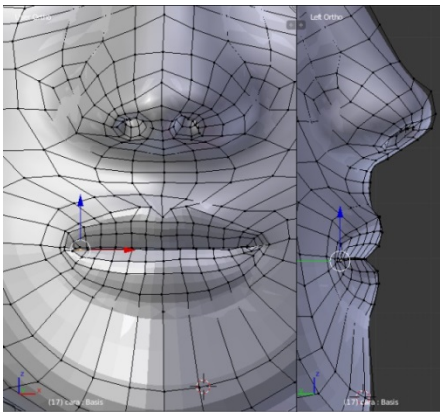


Figura 3.18 – Malla de la parte inferior del rostro: boca y mentón

3.3.1.3. Frente y cráneo

A partir de la malla obtenida con el rostro ya modelado, ahora hay que realizar la frente y el cráneo. En este caso no hay que modelar ninguna forma geométrica compleja mas allá que la forma de un cráneo manteniendo unas proporciones y dimensiones adecuadas. Siguiendo la forma de la imagen de la plantilla, la operación a realizar es una extrusión. Seleccionando la linea de vértices por encima del párpado superior, realizamos la extrusión de esta arista tantas veces como sea posible hasta completar la frente. Una vez que la frente ya este completada, se sigue extruyendo para mallar la zona del cráneo hasta llegar a la nuca.

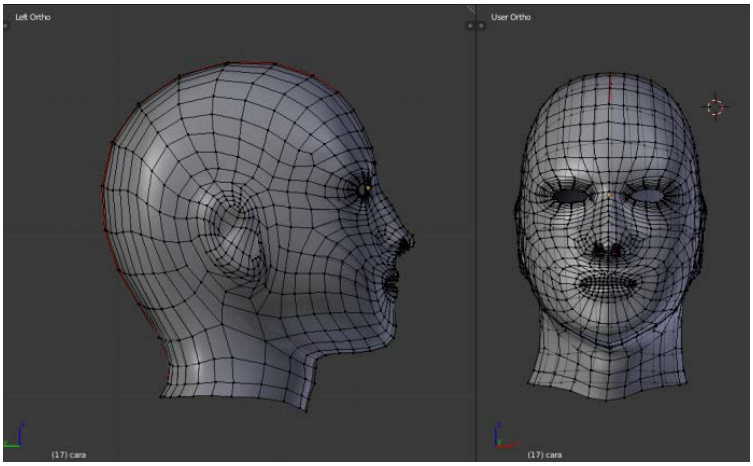


Figura 3.19 - Cabeza del avatar en perfil lateral y frontal

3.3.1.4. Interior de la boca

A fin de generar la zona interna de la boca se ha seguido el mismo proceso de extrusión, pero se han creado dos mallas distintas. Una malla que conforma la zona de los dientes superiores. Y otra malla que conforma la zona de los dientes inferiores. También se ha modelado la zona de las paredes interiores de la boca, y por último la lengua.

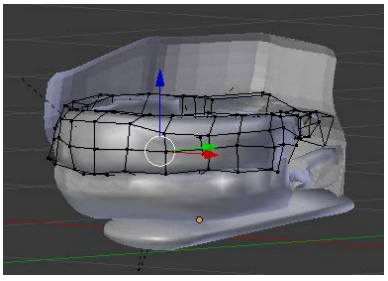


Figura 3.20 – Zona interior de la boca

3.3.2. Texturizado del avatar

3.3.2.1. Materiales y texturas en Blender

La aplicación de materiales y texturas en el diseño 3D es casi igual de importante que la propia forma del objeto. Sin este proceso todos los objetos que aparecen por pantalla lucirían grises y el efecto de la luz que se proyectase sobre la escena sería insustancial. Es por eso que proporcionar a los objetos una textura o material, hace que estos se vean más coloridos y realistas.

Existen algunos conceptos clave que es importante citar antes de seguir con el diseño del avatar. Uno de estos conceptos es el sistema de material y la textura en Blender. Material y textura no es lo mismo. El sistema de material permite al usuario modelar una gran variedad de materiales y cómo interactúan estos con la luz.

La diferencia entre el material y la textura es la siguiente: Un material define las propiedades ópticas de un objeto, por ejemplo el color o también el nivel de opacidad o transparencia. La textura por otro lado es un patrón que rompe con la apariencia uniforme de un material. Muy pocos objetos en el mundo real tienen una superficie lisa uniforme. Por el contrario, todos siguen algún tipo de patrón o combinación de patrones. Existen multitud de ejemplos de textura como la madera, la piel o el metal. Blender permite a las texturas que ejerzan influencia sobre los materiales de multitud de formas, como alterando sus colores. Asimismo también pueden combinarse varias texturas entre ellas para producir efectos originales. Para que una textura tenga efecto en el objeto, esta textura tiene que estar adjuntada al material del objeto. Se puede decir que la textura es uno de los varios parámetros que se pueden configurar en el material de un objeto.

Otro elemento interesante en los materiales son los *shaders*. Los *shaders* son unos elementos que determinan el cambio de apariencia del material según el ángulo de proyección de la luz. Hay una gran variedad de *shaders* y cada uno ofrece una propiedad óptica distinta: opacidad, transparencia, translucidez, cristalinidad, etc. La combinación de varios *shaders* puede producir resultados muy realistas en el diseño del objeto. Para el diseño de este avatar no se emplearán estos elementos, ya que se requiere de técnica y de tener bastante experiencia dentro del campo del modelado, iluminación y diseño de gráficos para obtener materiales extremadamente realistas. El texturizado y material es una disciplina extensa dentro de Blender y existen multitud de herramientas que dan muchas posibilidades de diseño.

En el siguiente apartado se va a comenzar con el texturizado. El objetivo es obtener el material que va a cubrir cada una de las partes del avatar. Este material se obtendrá a partir de unas texturas base que serán texturas de imágenes. Como el avatar ha sido modelado en distintos objetos independientes (ojos, interior de la boca, el rostro), el

texturizado se va realizar para cada uno de los objetos por separado, lo que facilita el proceso. Para la zona del rostro se va a realizar el texturizado a partir de dos métodos, por un lado mediante imágenes, y por otro lado mediante un texturizado manual. Después se visualizarán ambos resultados y se sacarán conclusiones al respecto.

3.3.2.2. Desenvoltura de malla 3D y mapeado UV

En el proceso del texturizado mediante imágenes, se asigna un grupo de pixeles de la imagen que va a servir como textura de las distintas formas poligonales de la malla que forman la superficie del objeto. Por esta razón, es necesario convertir los objetos tridimensionales a texturizar a un formato 2D, ya que las imágenes están en este formato. Una vez que se asigna la imagen al mapa 2D del objeto, al visualizar el objeto en tres dimensiones, la superficie que en un principio no tenía ninguna textura quedará cubierta por la textura imagen.

El primer paso es realizar la desenvoltura del objeto 3D en un mapa 2D de la malla del objeto. Antes de la desenvoltura, lo primero es marcar las costuras que el usuario considere en la figura 3D. Estas costuras son los puntos a partir de los cuales se desenvolverá la superficie 3D. Si no se marcan unas costuras y se desenvuelve el objeto directamente, puede haber puntos del objeto sobrepuestos en el mapa 2D, y será difícil editar y trabajar con el mapa 2D. A continuación, se seleccionan los vértices o aristas donde se desea realizar el corte. Una vez seleccionados, hay que marcar la opción *mark seam* dentro del apartado herramientas de borde. Esta opción generará las costuras en los puntos seleccionados. Las aristas donde estén las costuras se marcarán en color rojo. Para el caso del objeto “ojo” las costuras se han realizado en la parte trasera, en forma de cruz como se observa en la figura 3.21.

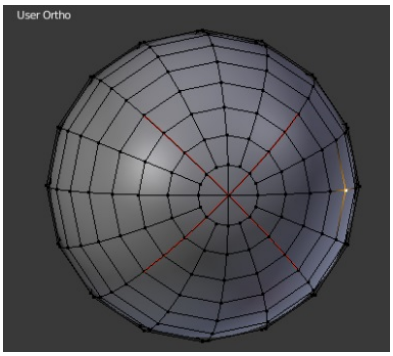


Figura 3.21 – Marcado de costuras en la parte trasera del ojo

Después del marcado de costuras hay que seleccionar todo el objeto y realizar la operación *Unwrap*. La operación *Unwrap* se encarga de desenvolver la malla 3D del objeto en un mapa 2D en función de las costuras que se hayan marcado. Para trabajar con los mapas 2D, está el editor *UV/image editor*.

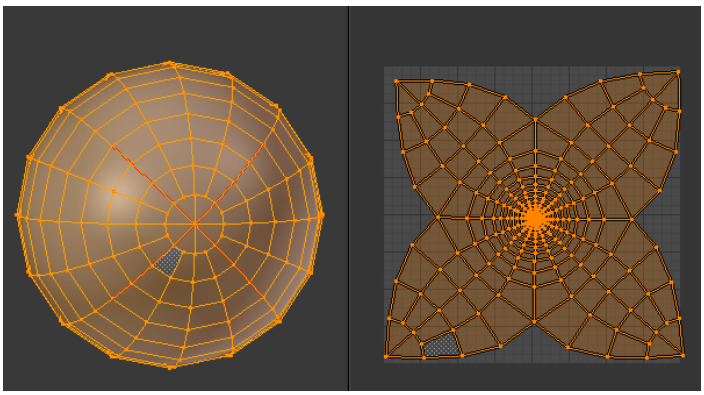


Figura 3.22 – Desenvoltura de la malla 3D en mapa 2D

3.3.2.3. Texturizado empleando imágenes de plantilla

Una vez desenvuelto el objeto en un mapa 2D, en el editor *UV/image* hay que cargar la imagen de plantilla que servirá como textura del objeto. El texturizado a partir de imágenes funciona de forma que teniendo el mapa 2D del objeto y la imagen, a cada forma poligonal del mapa, se le va a asignar como textura el fragmento de imagen que este sobre esa cara o forma poligonal.

Para este apartado es útil trabajar en doble ventana. Una ventana con el propio editor *UV/image* y otra ventana con el visor 3D. Por un lado en el editor UV se van ajustando los vértices del mapa sobre la imagen, y a la vez en el visor 3D, se visualiza si la textura esta aplicada bien sobre el objeto. Esto es porque en el mapa 2D no se aprecia si cada una de las caras contiene el fragmento de textura correcto, sin embargo en el visor 3D se puede visualizar correctamente.

En el caso de objeto “ojo”, la circunferencia central del mapa tiene que estar sobre la zona negra de la imagen, ya que ahí es donde se sitúa la pupila. Las tres circunferencias exteriores a esta, estarán sobre la zona marrón, grisácea que es el iris. La circunferencia exterior a estas sobre la parte blanca, que es la esclerótica del ojo. Así a cada forma poligonal de la malla 2D se le asigna un fragmento de pixeles de la imagen.

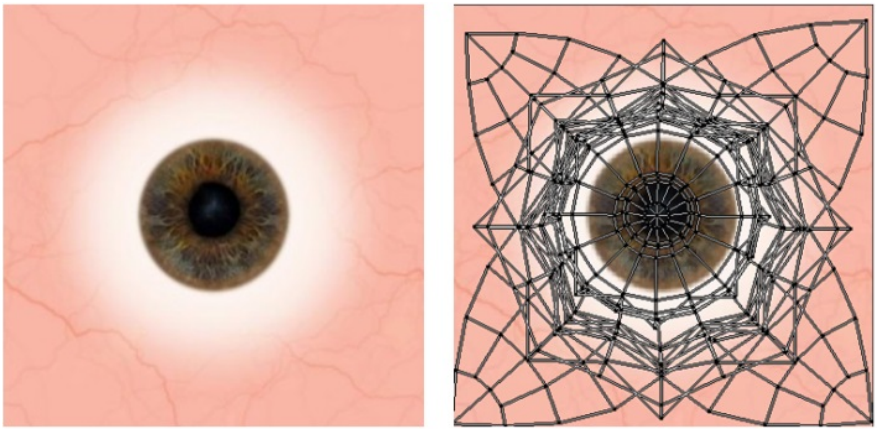


Figura 3.23 – Texturizado del ojo a partir de imagen de plantilla

Como resultado se obtiene ya el objeto texturizado. En el caso de que haya que texturizar varios objetos iguales, es más sencillo texturizar un objeto, y realizar la operación *duplicar objeto*, para obtener otro objeto idéntico al original. Una vez terminado, es importante guardar la textura como imagen en formato.png, porque de lo contrario al cerrar y abrir Blender aunque se haya guardado el modelo, la textura no se quedará guardada.

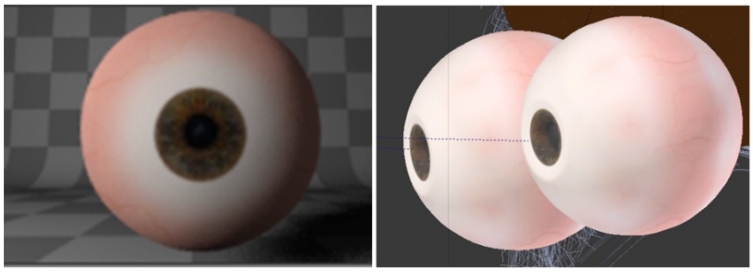


Figura 3.24 – Los dos ojos texturizados

El proceso para aplicar la textura al resto de objetos (dentadura, lengua y rostro) ha sido el mismo que en el caso de los ojos. Primero desenvolver el objeto 3D en un mapa 2D, asignar la zona de la imagen de textura a cada una de las caras del mapa, y finalmente guardar la textura en formato .png. El texturizado se ha realizado a partir de imágenes de plantilla. A continuación se muestran los resultados:

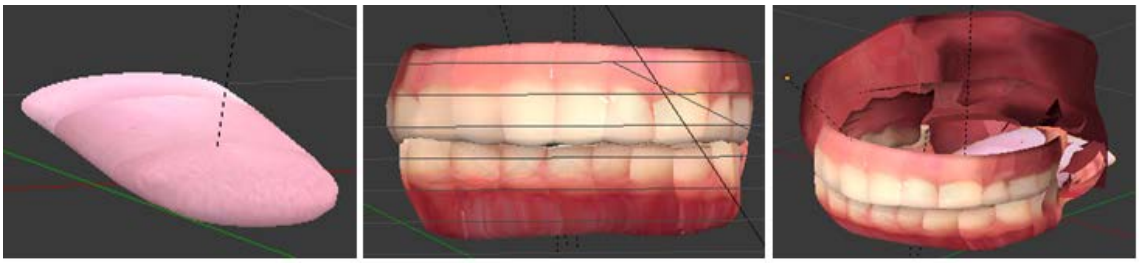


Figura 3.25 - Texturizado de lengua, dientes e interior de la boca

Figura 3.26 – Mapa 2D de la cabeza con la textura obtenida de la foto

3.3.2.4. Texturizado Manual

El texturizado manual consiste en generar una textura sobre el objeto a mano sin el uso de imágenes de plantilla. La ventaja de este método es que no hay que estar asignando a cada forma poligonal de la malla un fragmento de textura concreto como ocurría con el otro procedimiento. Este texturizado se realiza directamente con el objeto en 3D, y el uso del modo *Texture Paint*. Este modo ofrece herramientas tanto para texturizar como para esculpir el objeto en 3D. Esto permite generar defectos superficiales en la textura, como es el caso de la piel, y darle un aspecto más realista.

El primer paso para pintar la textura de la piel es emplear la herramienta *Brush Fill*. Esta herramienta permite cubrir todo la superficie de la malla de un color dentro de la escala RGB de colores. Como la piel no es uniforme en color y aspecto, se ha utilizado un paleta con un conjunto de colores cálidos entre amarillos-naranja-rosa, y se ha ido variando la

intensidad de los tres canales de cada uno de los colores para intentar dar con el color de la piel adecuado.

Después de obtener el color deseado, se ha empleado otra herramienta llamada *Texture Mask*, dentro del módulo *TexDraw*. Esta herramienta permite dibujar defectos en la superficie piel sin modificar el color de esta. Los defectos son producidos por una máscara. El proceso consiste en cargar una imagen que actúe de máscara, y después posicionar la máscara sobre la zona de la superficie deseada. Pasando el pincel sobre la máscara se consiguen trazar los defectos sobre la superficie cubierta por la máscara. Para el presente caso, se va a representar la textura de poros sobre la piel. Para ello se emplea una imagen con un texturizado de poros como la figura 3.27. Esta imagen se carga en el programa y actuará como máscara. Es importante que la imagen que sirva de máscara este en blanco y negro, para que la herramienta pueda establecer la discriminación entre el patrón a dibujar y el fondo.

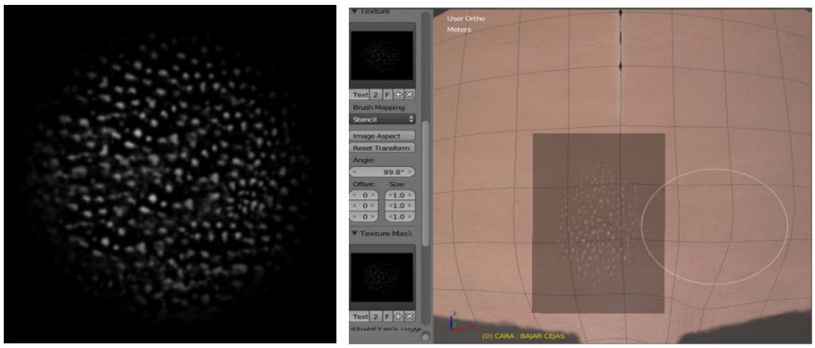


Figura 3.27 – Aplicación de la máscara de textura sobre la superficie del modelo

En la figura 3.28 se puede apreciar el efecto de los poros sobre la superficie de la piel.

Figura 3.28 – Efecto de la máscara de textura sobre la superficie

Una vez finalizado el proceso de añadir poros se ha pintado la zona de las cejas, pestañas, orejas y labios. Para pintar estas zonas se ha empleado *TexDraw*. *TexDraw* permite pintar manualmente con un pincel la superficie del objeto. Se puede editar tanto el color, tamaño e intensidad del pincel.

El texturizado manual puede ser más tedioso y llevar más tiempo que el texturizado a partir de una imagen ya que es un proceso manual, pero a la vez se pueden crear texturas muy originales y realistas si se tiene el conocimiento para saber combinar y emplear las herramientas que ofrece el editor. Una vez finalizada el texturizado, hay que guardar el mapa 2D de la textura como archivo de imagen para evitar perderla y tener que realizar todo el proceso de nuevo.

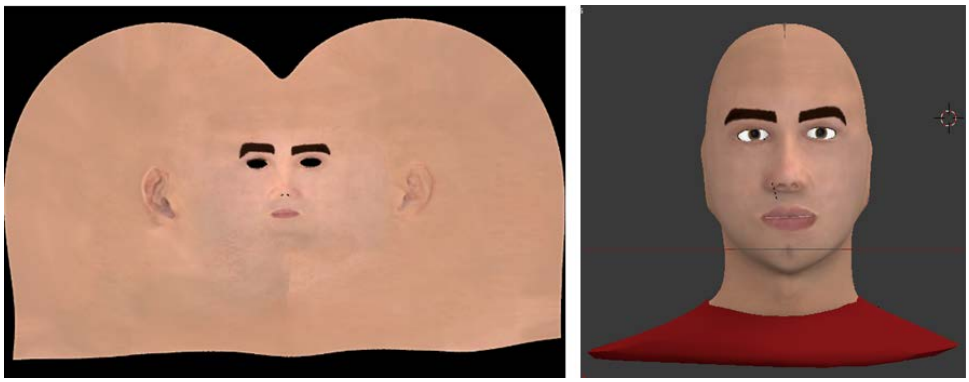


Figura 3.29 – Izquierda: Mapa 2D del avatar con la textura. Derecha: Avatar en 3D con las texturas añadidas

Una vez realizado el texturizado con ambos métodos, mediante imágenes de plantilla y manual, se pueden obtener algunas conclusiones. El texturizado manual permite obtener texturas personalizadas y de mayor calidad, sin embargo también requiere de tiempo y técnica en el estudio de modelaje 3D para conseguir ese realismo que compita con un rostro real. Esto es porque existen propiedades en los materiales donde se pueden editar y combinar muchos parámetros en relación a la iluminación como la translucidez, opacidad etc. Con el uso del texturizado a partir de imágenes se puede obtener una textura de una foto de un rostro real, pero la calidad de la textura en este caso depende menos del usuario y más de la calidad y resolución de la imagen que se va a emplear como textura. Para este trabajo en específico, se va emplear el modelo avatar con el texturizado manual para proyectarlo en el robot, simplemente por un tema de la estética, ya que el pelo facial en el robot queda extraño al estar el resto de la cabeza formada por un casco. No obstante, ambos resultados son válidos, y se puede emplear cualquiera de los avatares como modelo.

3.3.2.5. Asignar textura a material

Una vez texturizado todo el avatar, hay que asignar cada una de las texturas a su material correspondiente para que cuando se simule el avatar en el motor de juego, se puedan visualizar la texturas en el objeto 3D. Para asignar la textura a un material hay varios modos. Uno de ellos es mediante el editor de nodos. El editor de nodos permite un control casi ilimitado de los materiales del objeto. Permite añadir, conectar y combinar tantos nodos como el usuario considere, y de esta forma se pueden obtener infinidad de efectos distintos. La configuración de nodos para asignar la textura al material se muestra en la figura 3.30.

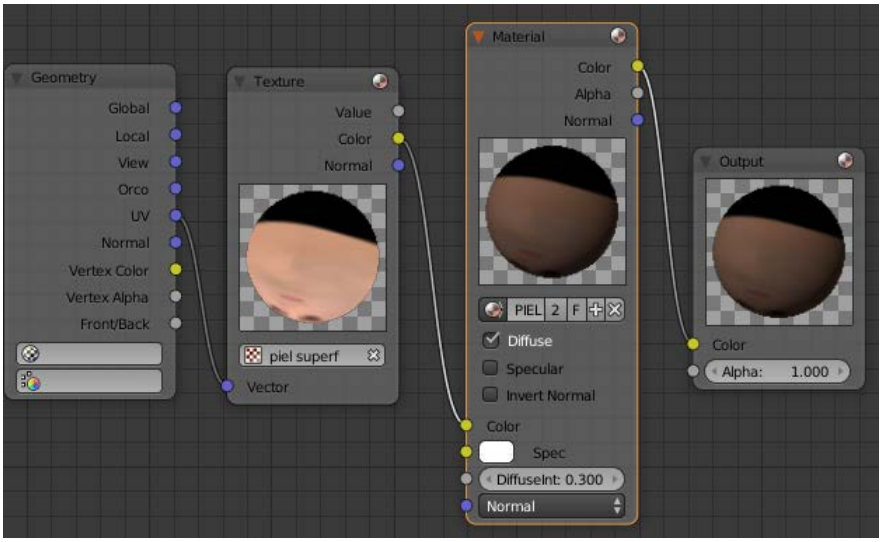


Figura 3.30 – Esquema de nodos para asignar la textura a un material del objeto

El nodo de entrada es *Geometry* que se va a conectar al nodo *Texture*. Esta conexión permite asociar a cada punto del mapa UV del objeto la textura diseñada. Después el nodo *Texture* se conecta al bloque *Material* a través de la entrada color para que el material adquiera el color de la textura. Finalmente se conecta el nodo *Material* al nodo de salida. Este mismo proceso también se va a realizar con el resto de texturas creadas: ojos, dientes, lengua y el fondo de la boca.

3.4. Animaciones faciales del avatar

El contenido de este apartado está dedicado a la animación del avatar virtual. El objetivo de la animación es implementar varias acciones sobre el avatar y controlar estas acciones en el motor de juego.

La metodología que se va a seguir para crear las animaciones es la siguiente:

1) Crear las formas clave.
2) Implementar una armadura de huesos que controlen las formas clave.
3) Insertar *keyframes* de los huesos y crear la animación de las unidades de acción en el editor de animación.
4) Mediante el editor lógico se controlará el sistema de animación.

Hay que mencionar que en un principio no se había planteado emplear las unidades de acción del FACS, sino que simplemente se habían generado las formas clave y su correspondiente hueso conforme al modelo que el autor consideraba que son los gestos faciales más naturales y realistas. Posteriormente, se ha decidido emplear las unidades de acción conforme el FACS para seguir un criterio universal a la hora de controlar las expresiones en el capítulo 4. Por esa razón, las formas clave y huesos implementados que aparecen en los apartados 3.4.1 y 3.4.2 son los que se implementaron en un inicio. Pero en el modelo final del avatar, sin borrar estas formas clave y huesos, y reutilizando algunos de ellos se han creado las unidades de acción del sistema de codificación mencionado (apartado 3.4.3). No obstante, se ha mantenido el contenido de los apartados 3.4.1 y 3.4.2 ya que ahí se explica el proceso para generar tanto las formas clave como los huesos.

3.4.1. Formas Clave

Las formas clave son un conjunto de formas creadas a partir de una forma base, para posteriormente crear una animación. La forma base es la forma original o primigenia que se le ha dado al objeto. En el caso del avatar, la forma base seria el rostro que se ha creado durante el diseño, pero sin expresión alguna. Las formas clave se generan deformando la forma base, modificando los vértices de la malla de objeto. Las formas clave que se crearán serán las distintas expresiones faciales.

Una característica de las formas clave es que no se pueden agregar o quitar vértices de la malla original. Solo se puede deformar la malla original, cambiando la posición de los vértices de la malla. Una ventaja de utilizar formas clave es que el usuario puede deformar la forma base como desee. La principal desventaja es que el modificar

manualmente cada uno de los puntos de la malla puede ser un trabajo tedioso. Pero más adelante se explicará un atajo para agilizar el proceso de la obtención de algunas formas clave específicas.

Con las formas clave se puede trabajar de dos modos: en modo absoluto o en modo relativo:

- Modo absoluto: La forma clave final es la interpolación de la forma clave anterior y la siguiente, dado un tiempo de evaluación.
- Modo relativo: Cada nueva forma clave se define en relación a una referencia. Normalmente la referencia es *Basis* o la forma base del objeto, pero también puede ser otra forma especificada. Este modo se utiliza principalmente para el diseño de músculos, expresiones faciales o el movimiento de articulaciones partiendo de una forma neutra. Este es el modo que se va a emplear para este avatar. Se crearán los distintos gestos partiendo de una forma base sin expresión alguna.

Existen dos parámetros principales en las formas clave:

- La base: Forma original del objeto. Sirve como referencia para crear nuevas formas clave.
- El valor: Es el peso de la mezcla entre la nueva forma clave y su forma base. Si el valor es 0, el 100% de influencia de la forma corresponde con la forma de referencia (la forma base). Si el valor es 1, el 100% de influencia de la forma corresponde con la forma clave creada. Si el valor es 0.5, el 50% de influencia de la forma, proviene de la forma de referencia y el otro 50% de influencia de la forma, proviene de la forma clave creada. Como las formas clave almacenan la posición de los vértices de la malla, para un valor 0.5, cada vértice P de la malla está a medio camino entre la posición de ese mismo vértice P en su forma base y la posición de ese mismo vértice P en la forma clave creada. En términos matemáticos, es una interpolación de la posición de los vértices de la malla a lo largo de un rango de valores, con un valor límite inferior 0 (forma base) y un valor límite superior, normalmente 1(formas clave).

A modo de ejemplo, en la figura 3.31 se muestra una forma clave cualquiera, en la que se han modificado ciertos vértices de los labios respecto de la forma original de la malla. De izquierda a derecha el valor o peso correspondiente de la forma clave: Izquierda, valor igual a cero. Centro, valor igual a 0.502. Derecha, valor igual a 1.

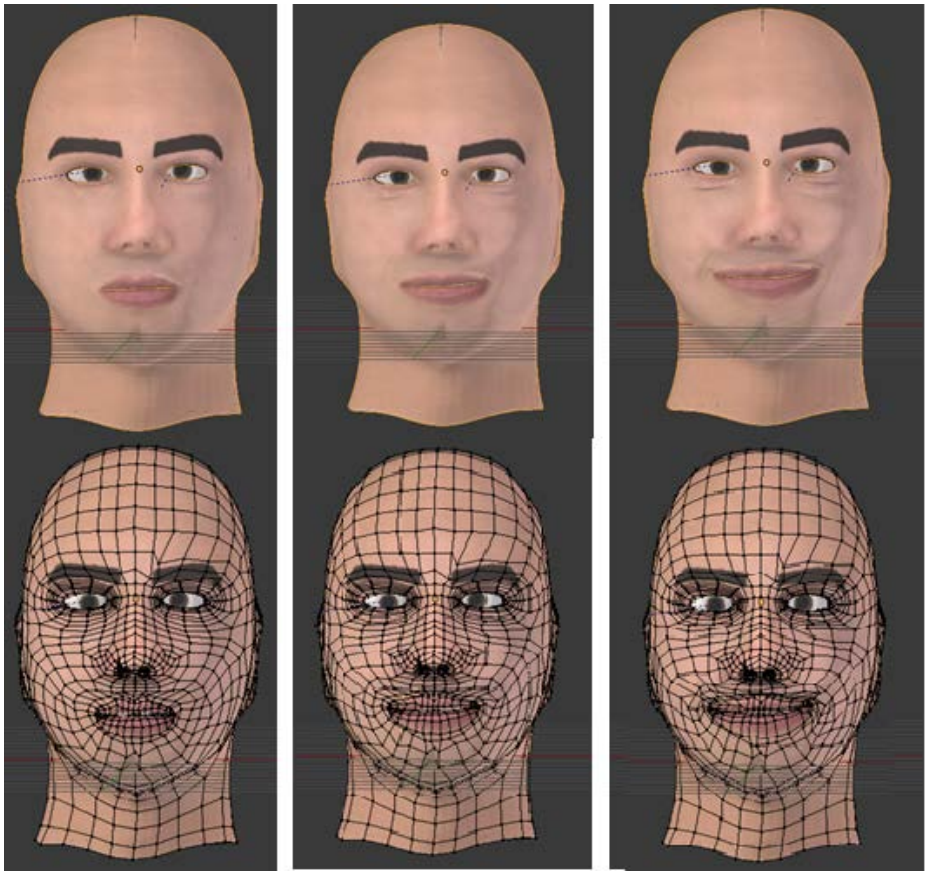


Figura 3.31 – Avatar en modo objeto (Arriba) y en modo edición (abajo) con distinto valor o peso de una forma clave

La ubicación donde se almacenan y se editan las formas claves creadas, es en el editor de propiedades, propiedades del objeto. En datos de objeto existe un apartado denominado *shape keys*. Dentro de este apartado, hay una tabla que es donde se almacenarán todas las formas clave que se han creado, además de la forma base. Se puede decir que cada forma clave es como un vector que guarda la ubicación de todos los vértices de la malla correspondiente a esa nueva forma creada. Si no se ha creado previamente ningún *shape key*, la tabla estará solamente con un único dato. Este dato se llama *basis*. *Basis* siempre aparece cuando se crea un objeto, y se actualiza siempre que modifiquemos la forma del objeto original y se tenga activa la opción *basis*.

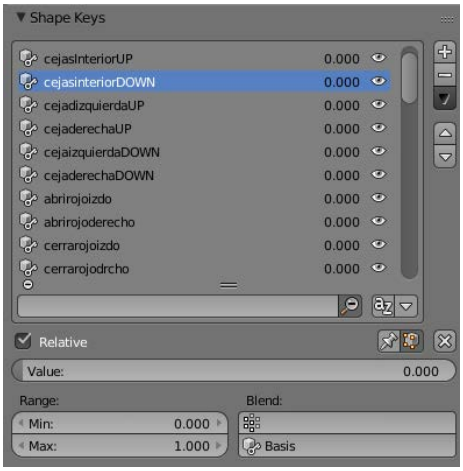


Figura 3.32 – Tabla de formas clave con algunas formas clave creadas

La barra de valor que existe debajo de la tabla en la figura 3.32, se puede manipular para observar en el editor 3D el estado de la malla de una forma clave, según el peso o valor que haya en la barra en ese momento. Junto a la barra hay un pequeño apartado llamado *Blend* que permite asociar un grupo de vértices de la malla a una forma de referencia. Más adelante, se describe el uso que se la ha dado a esta herramienta para formar algunas formas clave específicas.

3.4.1.1. Formas Clave Primarias

Se puede rellenar la tabla con la cantidad de formas clave que se deseen. Para añadir una nueva forma: En modo objeto, se presiona la *tecla* + y se da un nombre a la nueva forma clave para distinguirla del resto. En la tabla, aparecerá el nombre y un número que muestra el valor de la forma clave en ese momento. La forma base ya se explicó previamente que está en un valor cero siempre. Esto quiere decir que la influencia de una forma clave es nula, ya que no ha habido deformación de la forma original. Las formas clave se editarán con un peso igual a uno, o lo que es lo mismo cuando la influencia de la propia forma sea máxima sobre la forma original. Una vez que el valor está en uno, en modo edición y dentro de editor 3D, se modifica manualmente el objeto original deformando la malla, cambiando de posición los vértices que se deseen hasta conseguir la forma clave deseada.

Para después poder componer las animaciones y las expresiones faciales se han creado varias formas clave, desde el movimiento de las cejas, parpados, hasta el movimiento de la lengua y los labios en varias de posiciones. Llamaremos a estas formas clave, formas clave primarias ya que están hechas a partir de la forma base, deformando la forma original.

A continuación se muestra las formas clave principales:

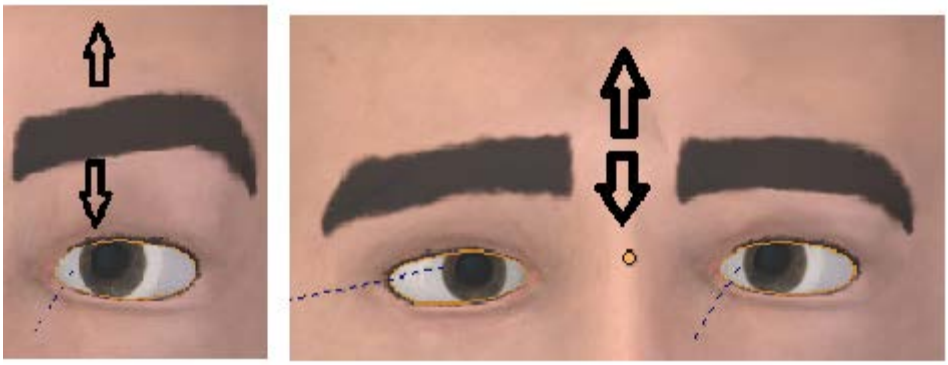


Figura 3.33 – Izquierda: Movimiento vertical de las cejas. Derecha: Movimiento del entrecejo arriba y abajo

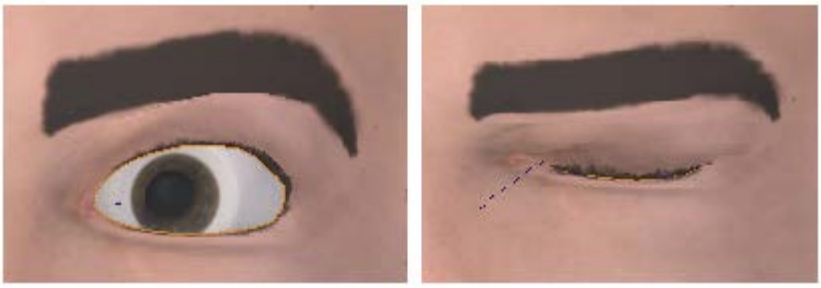


Figura 3.34 - Izquierda: Movimiento de los parpados. Acción: abrir ojo. Derecha: Movimiento de los parpados. Acción: cerrar ojo.

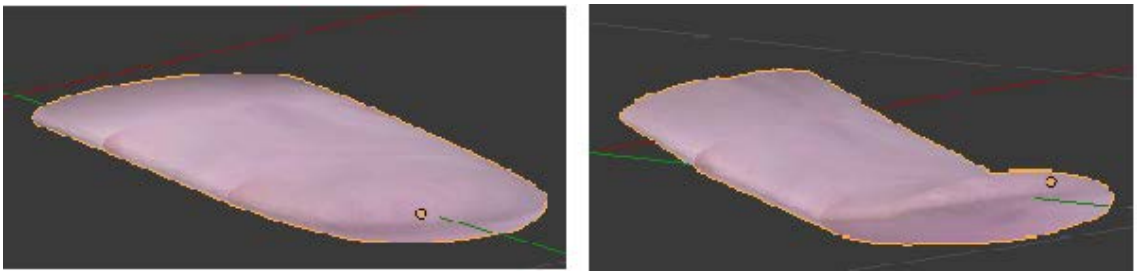


Figura 3.35- Izquierda: Lengua en posición neutra. Derecha: Movimiento de la lengua hacia arriba

Figura 3.36 - Izquierda: Subir el labio superior. Derecha: Bajar o apretar el labio superior.

Figura 3.37- Izquierda: Subir o apretar el labio inferior. Derecha: Bajar el labio inferior.

Figura 3.38 - Izquierda: Comisuras en diagonal superior. Acción: sonreír. Derecha: Comisuras en dirección diagonal inferior. Acción: tristeza.

Figura 3.39- Izquierda: Mover comisura en horizontal hacia afuera. Estirar labios. Derecha: Mover comisura en horizontal hacia dentro. Contraer labios.

3.4.1.2. Formas Clave Secundarias

Los movimientos labiales no tienen por qué ser únicamente simétricos. Cabe la posibilidad de poder mover parte de un labio y la otra parte mantenerlo en reposo. Es por eso que a continuación se va a ampliar el número de formas clave en la zona de los labios, para tener en cuenta estas nuevas situaciones.

Volver a crear manualmente nuevas formas clave a partir de la forma base, sería un trabajo muy largo y tedioso. Pero es posible utilizar un atajo para crear nuevas formas clave a partir de las ya implementadas. Ahora, a partir de estas formas clave primarias se van a crear unas formas clave secundarias que realicen el movimiento de una parte específica de los labios, no de todos los labios en conjunto.

Primero se ha de crear una copia de la forma clave primaria correspondiente. Se selecciona la forma clave primaria que se desea copiar, estando el valor o peso igual a 1 y se selecciona la opción *New shape from key*. Esta opción crea una nueva forma que es idéntica a la forma clave anterior.

El objetivo es implementar la nueva forma clave secundaria a partir de la forma clave primaria, pero solo en una mitad de los labios. Para ello, se va a ordenar que la forma clave primaria solo actué en un grupo específico de la malla del objeto. Existe un método para que la acción influya solo en ciertas partes de la malla, y este método consiste en la utilización del grupo de vértices o *Vertex group* y el editor *Weight Paint*.

Por un lado, el grupo de vértices se utiliza para etiquetar vértices pertenecientes a partes de la malla de un objeto. Por otro lado, el editor *Weight Paint* tiene como función la de poder asignar un peso o influencia a un grupo de vértices. *Weight Paint* es muy útil para varias aplicaciones. Una de estas aplicaciones, es cuando se quiere asociar un hueso a un objeto y la malla del objeto tiene que actuar conforme al movimiento del hueso. Otra aplicación que es la que concierne en este momento, es cuando se desea que la forma clave solo tenga influencia en un grupo específico de vértices de la malla (*Vertex group*).

En el editor de propiedades del objeto, en datos del objeto, hay un apartado dedicado a *Vertex group*. Ahí es donde se va a crear un nuevo grupo de vértices. Después, hay que ir al editor *Weight Paint* y asignar pesos a los vértices del grupo de vértices creado.

En el editor *Weight Paint* para asignar los pesos de los vértices se utiliza una especie de pincel para pintar los vértices de la malla. Las herramientas principales del pincel son:

- El valor o peso en un rango 0 a 1: Conforme el valor aumenta, la influencia sobre los vértices crece y los colores pasan de un color frio a un color caliente. Con valor 0, no existe influencia sobre el vértice, el color es azul oscuro. Con valor 1, existe influencia máxima sobre el vértice. El color es rojo.
- El radio del pincel: El Tamaño del pincel.

Como en el caso de los labios se desea que estos solo tengan influencia en cada una de las mitades por separado, se ha creado primero un grupo de vértices llamado "LadoDerecho", y mediante el editor *Weight Paint* se ha pintado de rojo la parte derecha de la cara, de forma que solo en la mitad derecha de la cara existirá influencia por parte de la forma clave. Por el contrario la mitad izquierda, va a quedar inmóvil en la posición original aunque la forma clave también actúe en esa zona de la cara. Posteriormente se ha creado otro grupo de vértices llamado "LadoIzquierdo", y la disposición de los pesos de los vértices es la opuesta. La zona izquierda es influenciable por la acción de la forma clave y la zona derecha quedará inmóvil ante la acción de las formas clave.

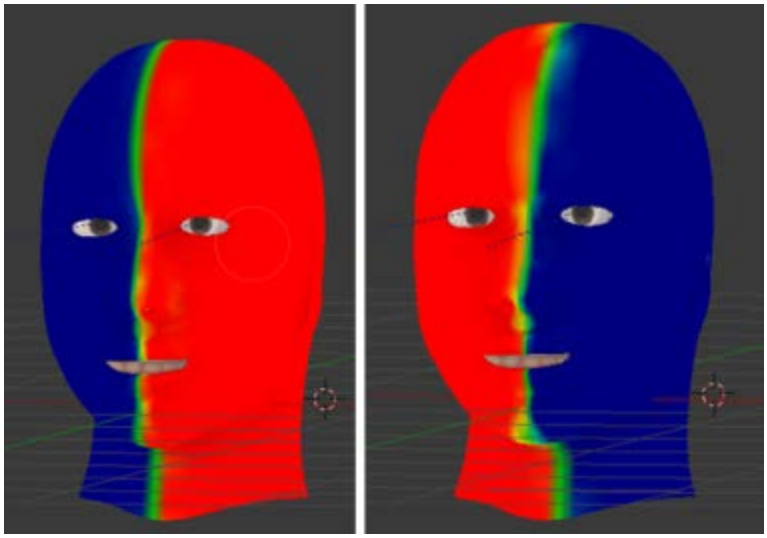


Figura 3.40 - Asociar pesos a grupos de vértices. Izquierda: Grupo de vértice: "LadoIzquierdo". Derecha: Grupo de vértices: "LadoDerecho".

El último paso, es volver a la tabla de los *shape keys*, y seleccionar la nueva forma clave creada, que era una copia de la forma clave primaria. A continuación, en el sub-apartado *Blend*, se va a especificar el modo de acción de la forma clave secundaria respecto a cada uno de los grupos de vértices correspondientes ("LadoIzquierdo", "LadoDerecho"). En la figura 3.41, se muestra un ejemplo del resultado de la creación de las formas clave secundarias a partir de una forma clave primaria.

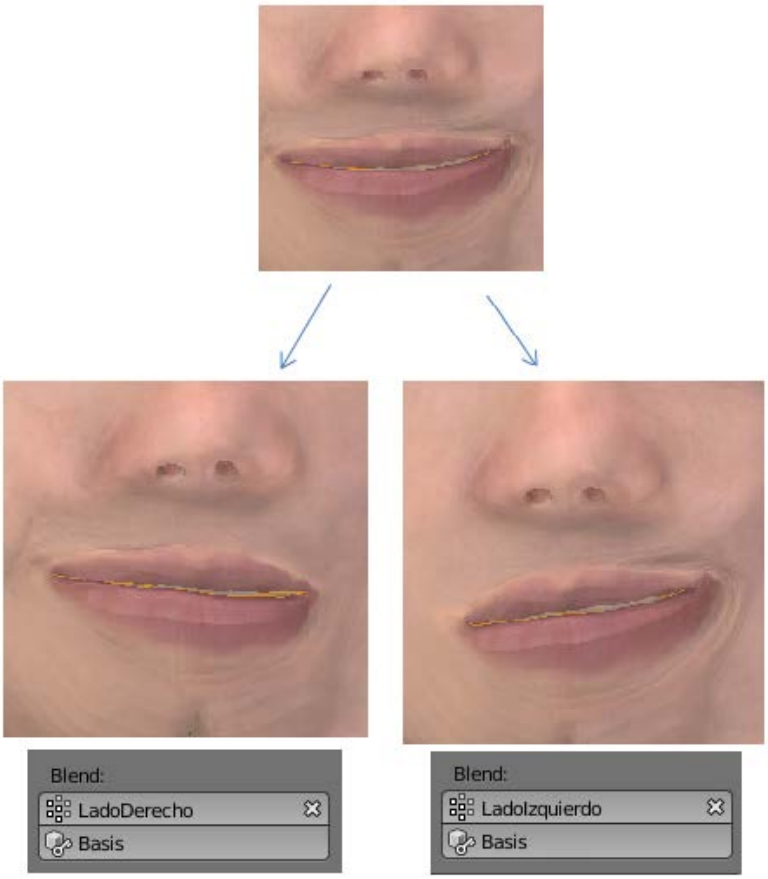


Figura 3.41 - Arriba: Forma clave primaria. Izquierda: Forma clave secundaria. Influencia solo en el lado derecho. Derecha: Forma clave secundaria. Influencia solo en el lado izquierdo

Siguiendo el mismo proceso, se obtiene el resto de las formas clave secundarias de los labios para poder representar cualquier acción o expresión facial.

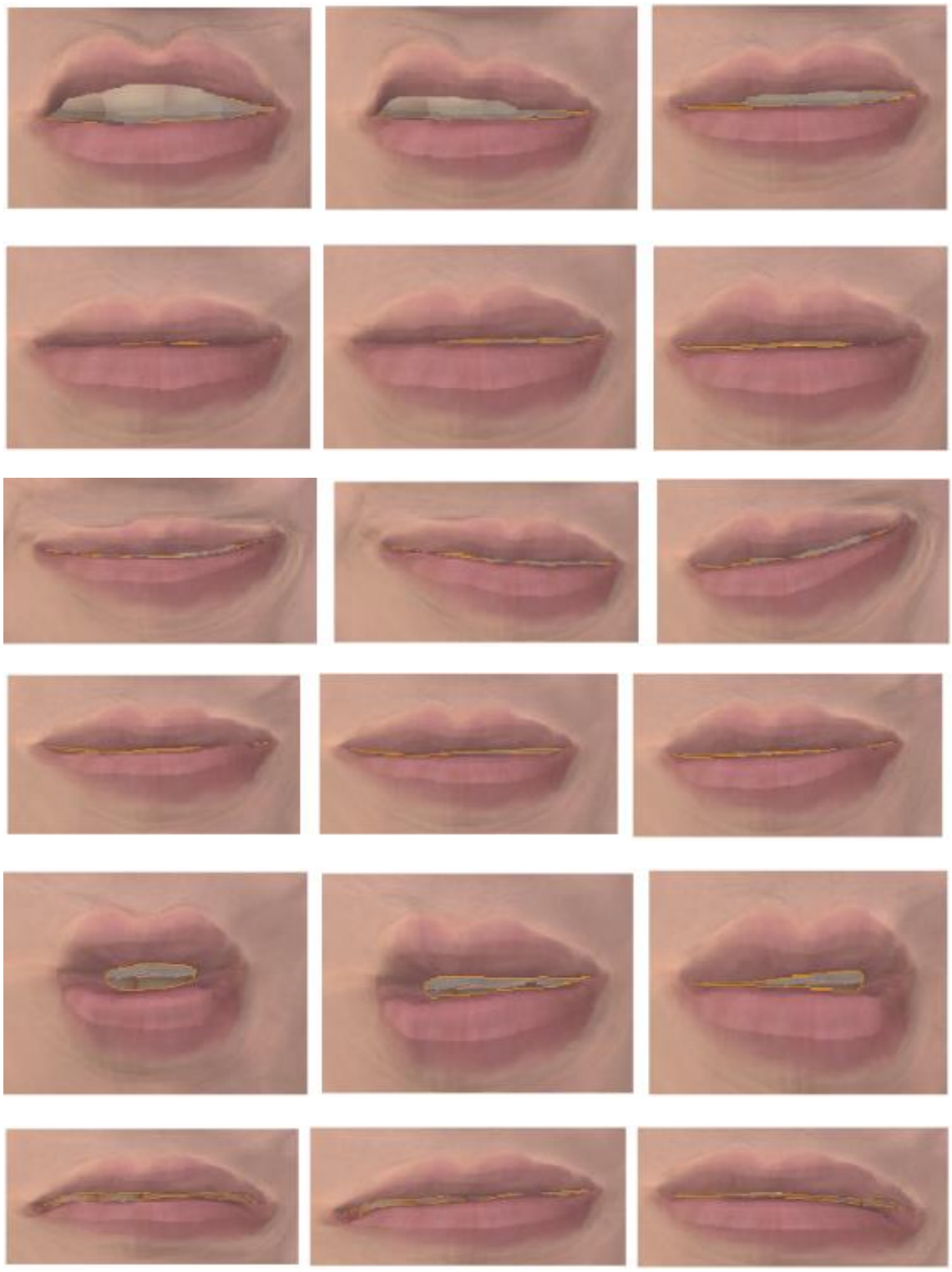


Figura 3.42 – Todas las Formas clave secundarias (centro y derecha) obtenidas a partir de su forma clave primaria (izquierda)

Algunos movimientos pueden resultar imposibles, para estos casos las formas clave se puede eliminar. Una vez implementadas las formas clave, hay que controlar su movimiento de algún modo. En este trabajo se hará uso de los huesos para controlar las formas clave. También se emplearán huesos para otras acciones que difícilmente se pueden implementar con formas clave como: controlar la dirección de la vista del avatar o abrir y cerrar la mandíbula. El procedimiento seguido se explica en el siguiente apartado.

3.4.2. Armadura y Huesos

Blender dispone de un grupo de elementos que permite controlar la animación de objetos que precisen de un movimiento articulado. Estos elementos lo forman la armadura y los huesos. Los huesos son elementos que se pueden implementar en distintas partes de un objeto para controlar tanto movimientos de traslación como de rotación del objeto. El aspecto que presentan los huesos en Blender es de forma octaédrica. Por su parte, la armadura es el sistema que conforma todo un conjunto de huesos. Se puede decir que la armadura equivale a la función que ejerce el esqueleto real sobre el cuerpo humano. Normalmente se agrupan huesos en una misma armadura si los huesos tienen una relación entre sí, pero no es necesario que un hueso tenga que estar unido o sea hijo de otro hueso para pertenecer a una misma armadura. Cuando un hueso está unido o conectado a otro hueso, el movimiento de un hueso va a influir en el otro y viceversa.

Un ejemplo de unión de huesos es en la figura 3.4.3. El movimiento de rotación en el eje Y de un hueso va a generar que el otro hueso también ejerza un movimiento de rotación sobre ese eje, y viceversa. Si no hay unión, no existirá influencia en el movimiento de un hueso sobre otro.

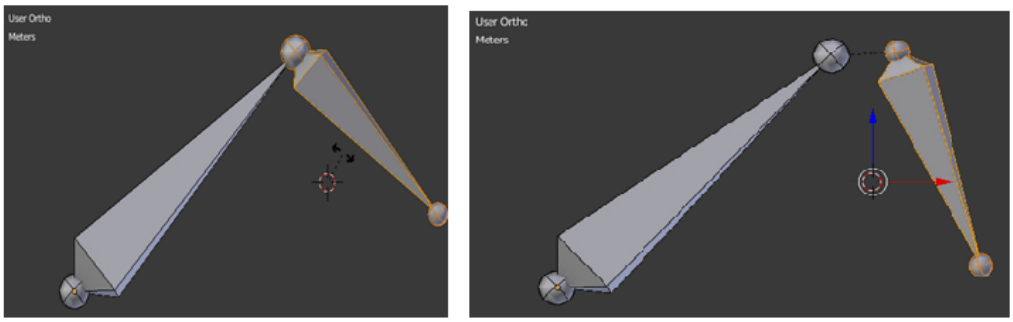


Figura 3.43 – Izquierda: Unión de huesos. Derecha: Huesos no unidos

Otra situación es cuando un hueso es hijo de otro hueso. La relación padre-hijo significa que el hueso hijo puede ejercer un movimiento propio, pero además también está influenciado por el movimiento del hueso padre. Pero no viceversa. El movimiento del hijo no afecta al padre.

Los métodos para trabajar con una armadura y huesos en el editor 3D son mediante la opción de modo objeto, modo edición y modo *pose*. En modo objeto, para editar el tamaño, ubicación y orientación de un hueso se emplean las mismas herramientas y comandos que se utilizan para un *mesh* u otra forma geométrica. Seleccionar el hueso y *tecla S* para escalar, *tecla G* para trasladar y *tecla R* para rotar. El modo edición es también igual que en los *mesh*. Se puede cambiar la posición de cualquier vértice del octaedro (hueso). Por último el modo *pose*, es un modo ya exclusivo para los huesos. En este modo se modela el movimiento o trayectoria que realizan los huesos, sus restricciones y la asignación y control de un hueso sobre un *mesh* u otro objeto.

Para este modelo, se va a crear una armadura llamada “DEFORMAR” que agrupe todos los huesos que estén implicados en la deformación del avatar, ya sean huesos que controlen a las formas clave, o huesos que deformen de manera directa el objeto.

3.4.2.1. Huesos de Deformación

Primero para añadir una armadura, hay que ir a modo objeto y seleccionar *añadir armadura*. Una vez que la armadura se ha creado, el primer hueso añadido es el hueso llamado "Cuello". Se ha añadido a mayores este hueso, aunque no sería tampoco necesario, ya que lo que se va a proyectar en la máscara al final es el rostro únicamente, no la parte del cuello. El hueso "Cuello" ejercerá la función de hueso padre para el resto de huesos siguientes. Unido a este, se ha añadido un hueso llamado "cabeza-objetivo". La función de este hueso es unir el cuello con la cabeza. A continuación el subsiguiente hueso hijo, es el hueso "Cabeza".

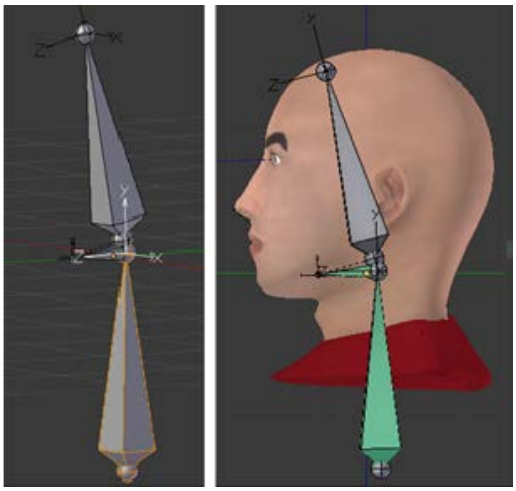


Figura 3.44 - Tres huesos que forman la columna vertebral del avatar

En este punto, el hueso "Cabeza", se va a reproducir generando el resto de huesos de la cara que van a controlar las formas clave o las facciones del rostro. En otras palabras, "Cabeza" será el hueso padre de todos los huesos de la cara. Si "Cabeza" no fuese el hueso padre, al mover la cabeza de posición, los huesos del rostro no cambiarían de posición en relación con la cabeza. La conexión entre los huesos padre-hijo, viene dada por una línea discontinua. Para facilitar la visualización de todos los huesos, estos se pueden colocar en distintas capas según la zona del rostro que vayan a controlar.

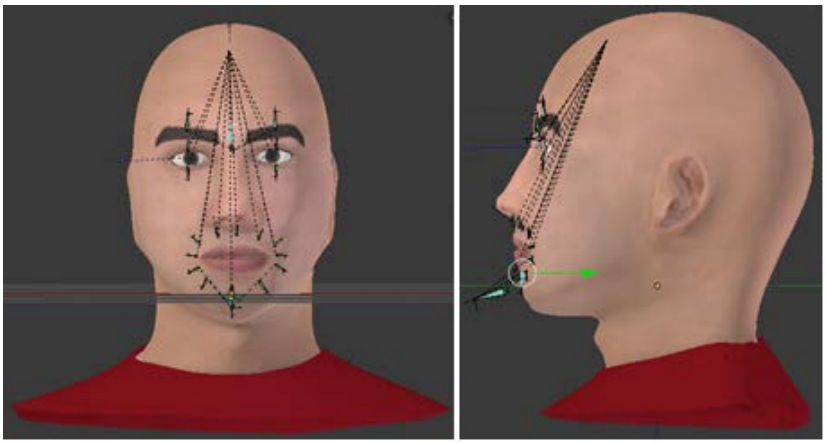


Figura 3.45 - El conjunto de todos los huesos de la cara en la misma capa. Todos los huesos están conectados al hueso padre "Cabeza"

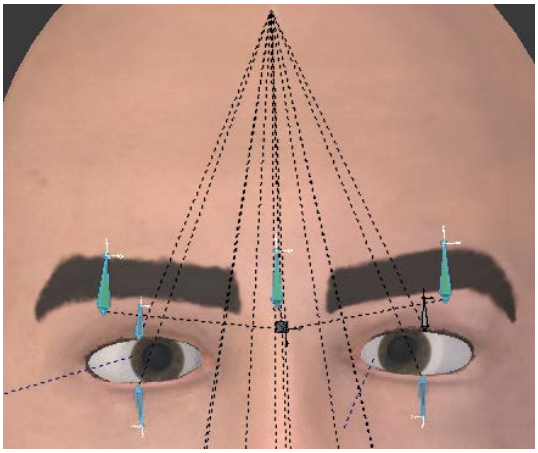


Figura 3.46 - Huesos correspondientes a las cejas, entrecejo y parpados inferior y superior de los ojos.

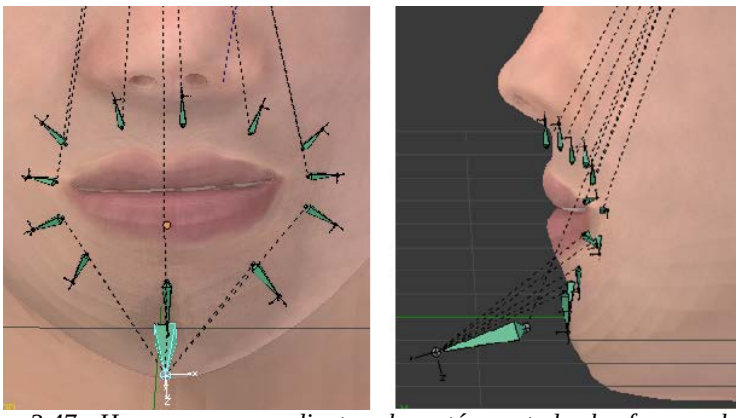


Figura 3.47 - Huesos correspondientes al mentón y a todas las formas clave de los labios.

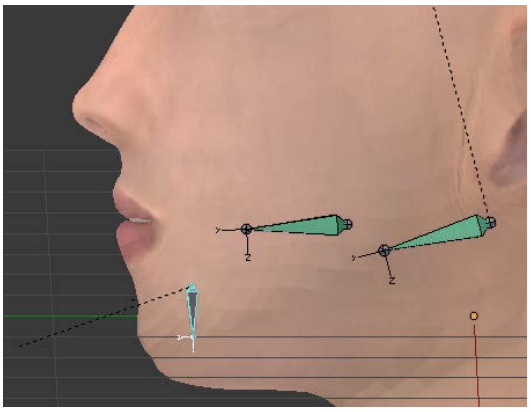


Figura 3.48 - Huesos correspondientes al movimiento de la mandíbula, dentadura y la lengua.

En este punto, todos los huesos de la armadura ya están ubicados en el editor 3D junto al avatar. Sin embargo, aún no existe una conexión entre el avatar y los huesos de la armadura. Para establecer la conexión objeto-armadura, hay que seleccionar la operación *set Parent to Armature Deform.*

El siguiente paso, es establecer la trayectoria o movimiento que va a seguir cada uno de los huesos, que después controlarán las formas clave. El objetivo es intentar que la trayectoria de cada uno de los huesos siga el mismo movimiento que el realizado por la forma clave que le corresponderá. Así, al implementar después los controladores o *drivers*, el control del hueso sobre la forma clave será más intuitivo. Blender permite trabajar tanto en coordenadas globales como locales. Para facilitar la configuración de la trayectoria de cada uno de los huesos se va a trabajar con las coordenadas locales del hueso. La colocación de todos los huesos del rostro se ha dispuesto de forma que la trayectoria de estos sea un movimiento de traslación sobre el eje Y local de cada hueso.

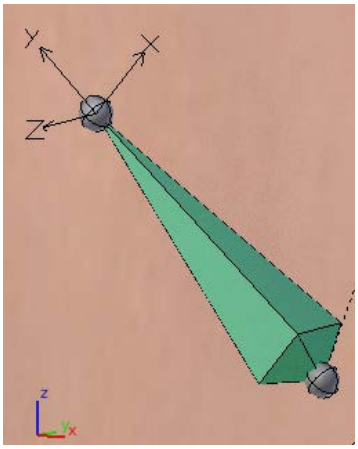


Figura 3.49 - Coordenadas globales (imagen abajo-izquierda) y coordenadas locales del hueso

Si el hueso se va mover solo en el eje Y, hay que bloquear la traslación en el resto de ejes. Además, se va a añadir una restricción de movimiento, para evitar que el hueso se mueva indefinidamente a lo largo del eje Y. Para añadir restricciones, ir a propiedades del objeto hueso, y seleccionar *Bone Constraints.*

El último paso, es controlar con los huesos la deformación del avatar. Los huesos del rostro controlarán las formas clave de la cara y el resto de huesos algunos movimientos especiales.

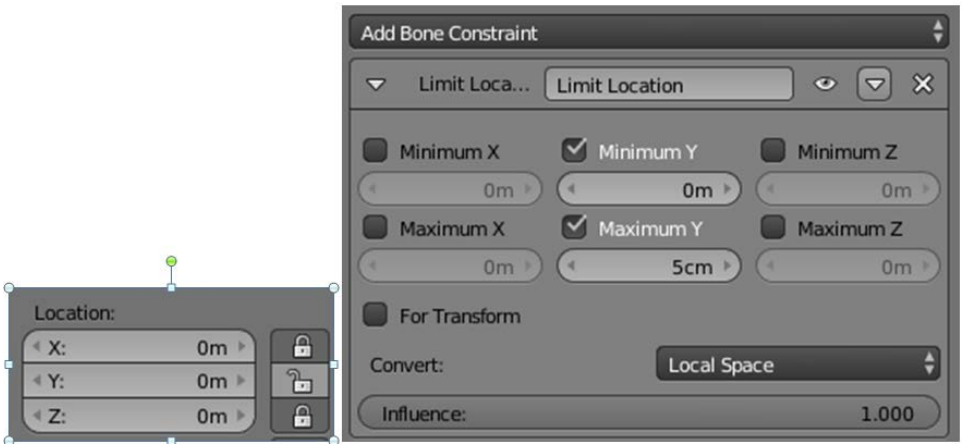


Figura 3.50 - Izquierda: bloquear traslación en los ejes X y Z locales del hueso. Derecha: Restricción de la traslación en eje Y local en un rango de valores

3.4.2.1.1. Huesos para controlar formas clave

En lo que respecta al control de las formas clave se va utilizar los *drivers* o controladores. Un *driver* o controlador es un mecanismo que sirve para controlar los valores de la propiedad de un objeto a partir de una función, expresión matemática o variable.

En el caso presente, la propiedad del objeto (avatar) a controlar, es el peso de las formas clave. El valor del peso representa la traslación de los vértices de cada una de las formas clave, con respecto a la forma original. Y la variable que va a controlar esta propiedad es una variable de tipo *Transform Channel.* La variable *Transform Channel* utiliza un objeto o hueso como variable, para que la propiedad controlada cambie conforme cambia la variable hasta llegar al valor de la transformación final del objeto o hueso. En este caso la variable que controla, es la trayectoria del hueso en el eje Y local.

Para asignar un *driver* a la forma clave, se selecciona la forma clave deseada de la tabla con el peso igual a 1, y sobre la misma barra del peso seleccionar *Add driver.* En el editor de gráficos, dentro del apartado *drivers* se va a editar el controlador. Lo primero es seleccionar el tipo de controlador, en este caso va a ser *Transform Channel.* Se selecciona

el hueso que va a controlar a la forma clave, y el tipo de movimiento del hueso. En todos los huesos se ha especificado que el movimiento es de traslación en el eje Y local.

Después hay que ir a las curvas F. La grafica de las curvas F representan la interpolación entre dos propiedades. El eje vertical de la gráfica, va a representar la propiedad controlada. Esta propiedad es el peso de las formas clave. En el apartado de las formas clave se definió un valor de peso mínimo igual a 0, donde la forma clave no tenía influencia sobre el objeto, y un peso máximo igual a 1, donde la forma clave tenia influencia máxima en el objeto. El rango de pesos entre 0 y 1 muestran el paso de la forma original a la forma clave.

En el eje horizontal de la gráfica, se representa la variable que controla a dicha propiedad. La variable es la trayectoria del hueso que previamente se ha especificado como movimiento de traslación en el eje Y local.

En la figura 3.51, se muestra un ejemplo de la curva F de los *drivers* para el control de una de las formas clave. Se desea controlar la forma clave de la comisura izquierda del labio a partir de su hueso correspondiente. El hueso tendrá un movimiento en el eje Y local restringido desde 0 cm hasta un valor máximo de 5cm. Cuando el hueso está en reposo, en la posición 0 cm (eje X) de la gráfica, el peso de la forma clave es igual a 0. No hay forma clave. Cuando el hueso se desplace a su valor máximo 0.05m, el peso de la forma clave será igual a 1, es decir, la forma clave quedará representada. La relación va ser proporcional. Mientras el hueso se esté desplazando desde su posición mínimo hasta su posición máxima, el peso o influencia de la forma clave ira creciendo proporcionalmente hasta llegar al peso máximo.

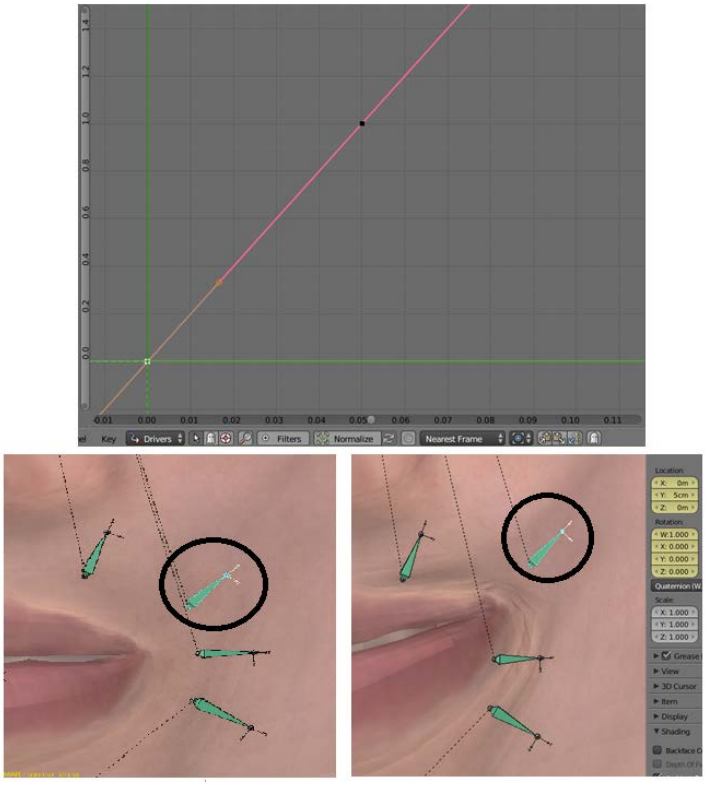


Figura 3.51 - Arriba: Gráfica del editor de drivers. Abajo-Izquierda: hueso en reposo. Forma base. Abajo-derecha: Hueso desplazado. Se genera la forma clave.

Con este procedimiento, las formas clave se generarán solo cuando se mueva su hueso correspondiente. Algunos huesos controlarán incluso dos formas clave. Se ha seguido el mismo proceso para controlar el resto de formas clave con los huesos restantes del rostro: labios, parpados superior e inferior, cejas y entrecejo.

3.4.2.1.2. Huesos para movimientos especiales

Algunos movimientos de la cara resultan muy complicados de implementar utilizando formas clave, además de que los movimientos resultantes no quedan muy naturales. Para estos casos, el hueso se encargará directamente de controlar partes de la malla, sin el uso de las formas clave.

Uno de estos movimientos es el de la mandíbula, para abrir y cerrar la boca. El interior de la boca humana está formado por el hueso maxilar superior junto con la arcada dental superior, y el hueso maxilar inferior (mandíbula), junto con la arcada dental inferior. Ambos maxilares están unidos en la articulación temporomandibular. Esta articulación permite el movimiento para abrir y cerrar la boca. Mientras que el maxilar superior se mantiene fijo junto con la arcada dental superior, la mandíbula o maxilar inferior es la

que se mueve arriba y abajo. Los dientes de la arcada dental inferior se mueven conforme al movimiento de la mandíbula ya que están unidos al propio hueso.

Para controlar el movimiento de la mandíbula en el avatar, se va a emplear un hueso llamado “mentón” que actuará como hueso padre que controle el movimiento arriba-abajo de la mandíbula. El movimiento del “mentón” se ha restringido a un movimiento de traslación arriba-abajo, en su eje Z local, de 0 a 12cm. Al mover la mandíbula hacia abajo, por inercia, la boca siempre se va a abrir y el labio inferior se desplazará hacia abajo, mientras que el labio superior se mantendrá fijo. Teniendo en cuenta este razonamiento, el hueso padre “mentón” se conectará a los huesos que forman el labio inferior, que serán sus huesos hijo. Los huesos hijo realizarán el mismo movimiento que el padre, además de los suyos propios, si tienen. De esta forma, al desplazar hacia abajo el hueso mentón (padre), los huesos hijo del labio inferior se van a mover también hacia abajo. Ahora los huesos del labio inferior ya pueden moverse independientemente de si la boca está abierta o cerrada.

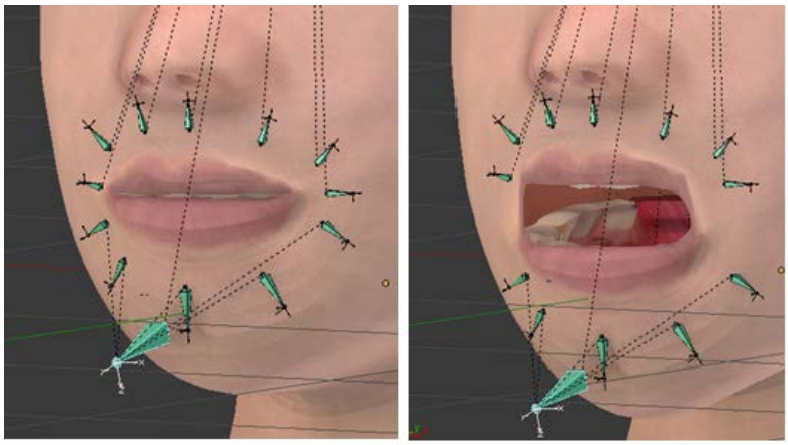


Figura 3.52 - Izquierda: Huesos en reposo. Derecha: Al mover el hueso “mentón” se va a abrir la boca. Los huesos del labio inferior se van a mover conforme a su hueso padre “mentón”.

Si se mueve la mandíbula, además del labio inferior, los dientes inferiores también han de moverse arriba-abajo. Para ello se ha creado otro hueso llamado “dientes_inferiores” que será otro hijo del hueso padre “mentón”. Al mover el hueso mentón hacia abajo, los dientes inferiores también bajarán.

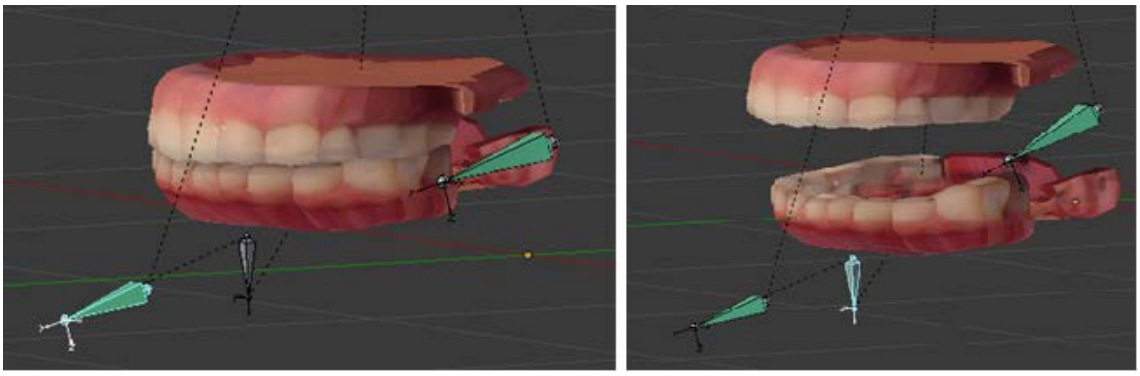


Figura 3.53 – Izquierda: Huesos en reposo. Derecha: Los dientes bajan conforme al movimiento de la mandíbula.

Además del labio inferior y dientes inferiores, el movimiento de la mandíbula genera también movimiento en el rostro, en la zona de los pómulos, mejillas y el propio mentón. Para ello se ha creado el hueso "Mandíbula" que moverá todo el tren inferior del rostro del avatar. Como se ha citado previamente, el hueso "mentón" controla el movimiento de la mandíbula, pero el que mueve la mandíbula en sí, va a ser el hueso "Mandíbula". Para seleccionar la zona de influencia de la malla en la que va actuar el hueso, se va a emplear la herramienta *Weight Paint*. La zona roja pintada es la zona de máxima influencia o la zona de máximo movimiento. La zona azul oscuro es la zona de la malla con cero influencias por parte del hueso.

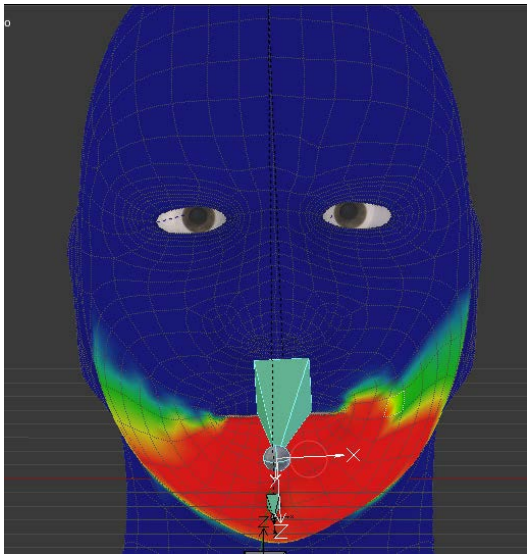


Figura 3.54 - Herramienta Weight Paint para asignar un peso a los vértices de la malla que conforman la zona maxilar inferior.

Para mover el hueso "Mandíbula" no se va a emplear la conexión padre-hijo con el hueso "mentón" como se ha hecho antes con el labio y dientes. Esto se debe a que el hueso hijo, copia exactamente el movimiento del padre. Pero en el caso actual, el movimiento de "Mandíbula" no es el mismo que el del hueso "mentón". El hueso "Mandíbula" tendrá un movimiento de rotación en el eje X local del hueso, mientras que el hueso mentón tiene un movimiento de traslación en el eje Z local del hueso.

La forma correcta de controlar "Mandíbula" mediante el "mentón" es utilizando una restricción en el hueso llamado *Track To*. *Track To* restringe el movimiento de un hueso, en este caso "Mandíbula" mediante el seguimiento de otro hueso que será "mentón". De este modo el hueso "Mandíbula" va a rotar hasta que el movimiento de "mentón" (movimiento de traslación vertical) finalice.

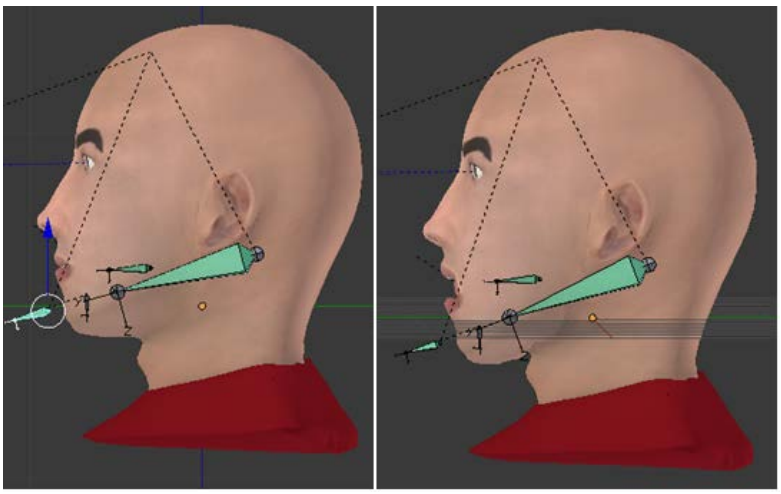


Figura 3.55 – Movimiento de rotación de la mandíbula restringida por el hueso mentón.

Una vez finalizado el control de la mandíbula y los dientes, otro de los movimientos especiales que se ha implementado, es el de la vista del avatar hacia distintos puntos del espacio. Para este movimiento, tanto el iris como la pupila de ambos ojos van a estar orientados es una misma dirección. Para controlar la dirección de visualización se empleará un objetivo o punto de seguimiento.

Antes de implementar el objetivo, hay que asegurarse que dentro del sistema de coordenadas local del objeto ojo, ambos ojos, tengan ubicadas los ejes de coordenadas en la misma dirección. Esto es importante ya que se desea que la acción de "ver" sea coordinada. Para visualizar el sistema de coordenadas de un objeto se presiona la opción *axis* dentro del editor de propiedades del objeto. En el modelo, se ha colocado el eje Y local de cada uno de los ojos en dirección vertical, y el eje Z local de cada uno de los dos ojos, apuntando hacia la pupila, es decir en la dirección de seguimiento del objetivo.

Una vez configurada la dirección del movimiento de los ojos, hay que añadir un objetivo para visualizarlo y seguirlo. La figura utilizada para representarlo será un hueso llamado *"eyetrack_bone"*, que controlará la dirección de movimiento de las pupilas, mediante el cambio de posición del propio hueso.

Para establecer la relación entre el hueso y los ojos, hay que añadir una restricción de movimiento a cada uno de los ojos. Las restricciones se modifican dentro del editor de propiedades del objeto, en el apartado restricciones. Al igual que para el movimiento de la mandíbula, también se ha empleado la restricción *Track To.* Esta restricción provoca que el movimiento de los ojos este controlado por la posición del hueso objetivo. Primeramente se especifica el objetivo a seguir, que es el hueso "*eyetrack_bone"* La siguiente opción a configurar es *Head/Tail*, que permite establecer el punto exacto de seguimiento dentro de la geometría del hueso. Los huesos en Blender tienen como punto inicial y final, una cabeza y una cola. Si el valor *Head/Tail* es cero, el punto de seguimiento irá a la cabeza del hueso. Si el valor es uno, el punto de seguimiento, irá a la cola del hueso. En este modelo, para cada ojo, se ha ido modificando el valor *Head\Tail* hasta conseguir que las dos pupilas estén apuntando en la misma dirección. Por último, se especifica el eje de coordenadas que va a seguir al objetivo, en este caso el eje Z, que es el eje hacia donde se desea que apunten las pupilas.

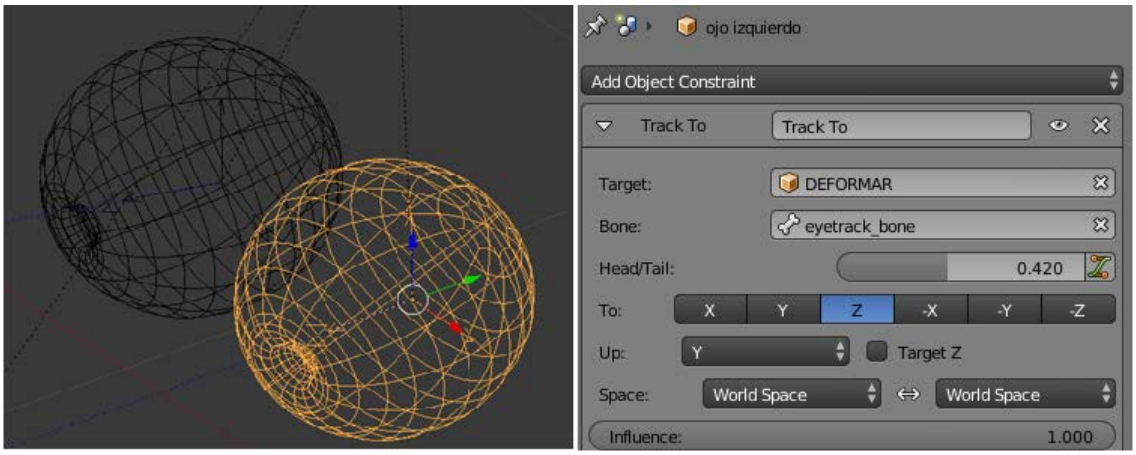


Figura 3.56 - Izquierda: Dirección de las pupilas de ambos ojos en el eje Z local. Derecha: Restricción "Track To" de movimiento del ojo.

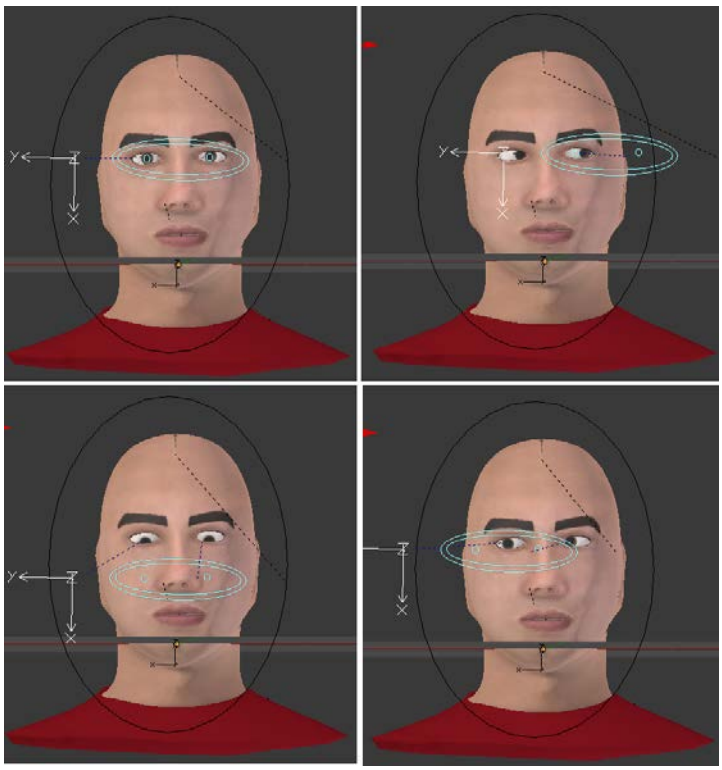


Figura 3.57 - Seguimiento de la vista al objetivo, en distintas posiciones.

El último de los movimientos especiales es el del cuello, aunque para este trabajo no es necesario implementar este movimiento en el modelo. El control del giro del cuello se realiza de forma similar al movimiento de las pupilas, con el uso de un punto de seguimiento que controle la dirección de giro del cuello. El hueso “cabeza-objetivo” es el hueso que se encuentra unido al cuello y será el hueso que ejerza la rotación, conforme al seguimiento del punto de referencia. El punto de referencia u objetivo es el hueso denominado “movimiento_cuello”. El movimiento del hueso “cabeza-objetivo” estará restringido o controlado según la posición de “movimiento_cuello” y unos ángulos que delimitarán el giro máximo de la articulación. El rango de giro del cuello estará comprendido entre unos valores de -20 a 20 grados, tanto en el eje X de rotación, como en el eje Z de rotación. La rotación en el eje Y está bloqueado, ya que la dirección del hueso “cabeza-objetivo” se encuentra sobre el propio eje Y, y resultaría un giro de cuello imposible.

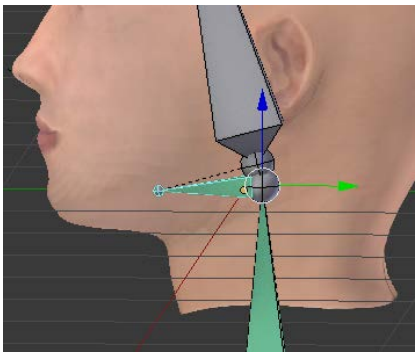


Figura 3.58 – Hueso "cabeza-objetivo" hace de unión entre el cuello y la cabeza

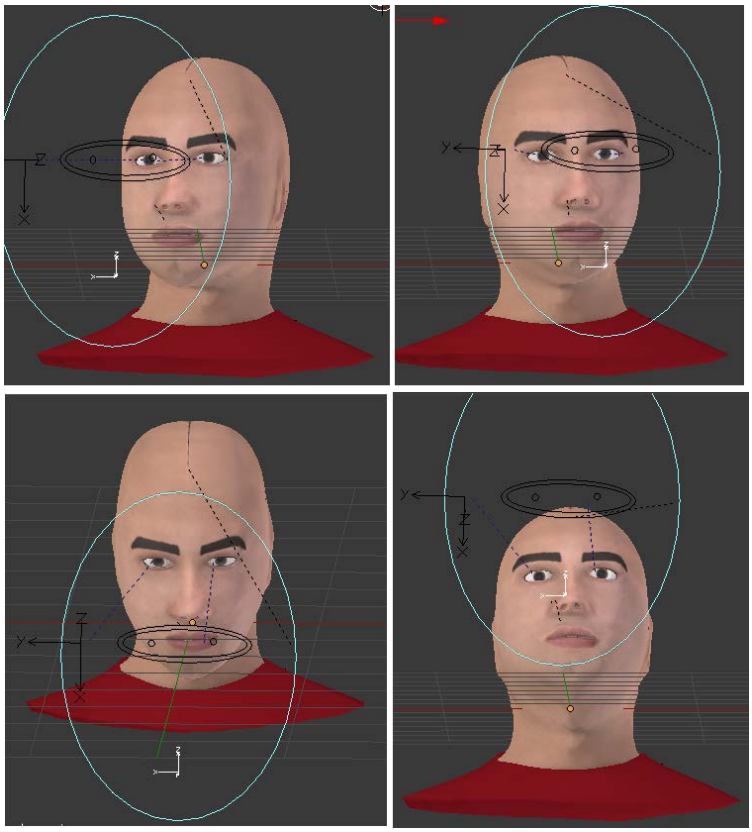


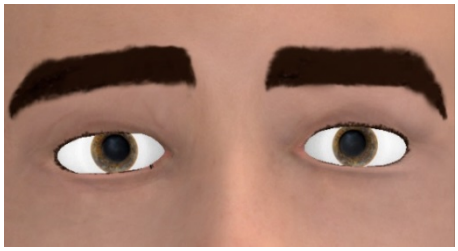
Figura 3.59 - Movimientos del cuello según la posición del objetivo

En el modelo tanto la armadura de huesos como el avatar se han colocado en distintas capas. Se puede seleccionar que más de una capa este visible a la vez. Para cambiar de capa hay que seleccionar el objeto deseado, *tecla M* y elegir la capa en la que almacenar el objeto o hueso.

3.4.3. Sistema FACS para las unidades de acción

Como se ha mencionado al comienzo del capítulo, en un principio se planteó implementar los huesos explicados según el criterio del autor. Aunque con esos huesos también se pueden representar cualquier gesto facial, más tarde se decidió emplear el sistema FACS, ya que es un método de codificación estándar y universal. También es más sencillo identificar cada movimiento ya que se le asigna una etiqueta (número) a cada unidad de acción. Además, cuando otros usuarios manipulen los movimientos del avatar, estos puedan seguir un procedimiento estandarizado como es el FACS. Algunos de los huesos anteriores se han reutilizado, como la abertura de la mandíbula, y el resto de huesos se han ubicado en otra capa distinta.

La unidad de acción representa cada micro-movimiento del rostro. Por lo tanto en Blender, cada unidad de acción se puede considerar como la suma de una forma clave más su hueso correspondiente. En las tablas 5 y 6 se muestran las unidades de acción según el FACS que se han añadido al avatar. Estas unidades se muestran en su máxima intensidad.



AU1-> Levantar interior de las cejas

AU2 -> Levantar exterior de las cejas

AU4 -> Bajar cejas

AU5-> Levantar párpados superior

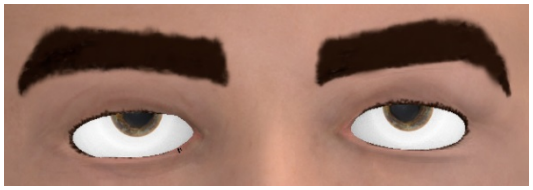
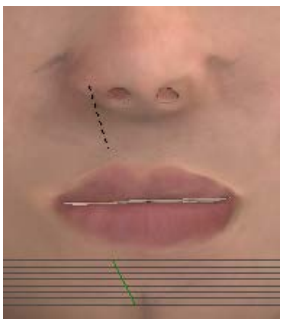
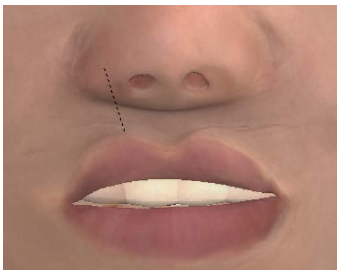
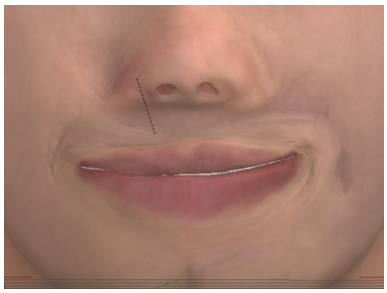
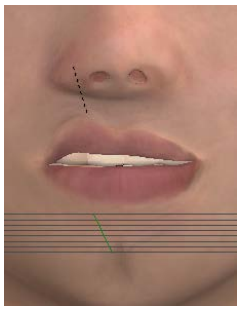
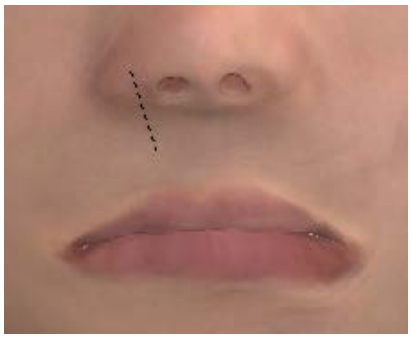
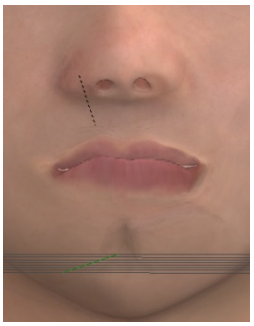
 **AU6 -> Levantar mejillas=Arruga de párpados**	 **AU7 -> Apretar parpados**
 **AU45 -> Parpadeo**	 **AU61 -> Vista a la izquierda**
 **AU62 -> Vista a la derecha**	 **AU63 -> Vista arriba**
 **AU64 -> Vista abajo**	

Tabla 5 – Codificación FACS de la parte superior del rostro del avatar

<table>
<tr>
<td> 
AU9 -> Arrugar la nariz</td>
<td> 
AU10 -> Levantamiento del labio superior</td>
</tr>
<tr>
<td> 
AU12 -> Tiramiento labial esquinal</td>
<td> 
AU13 -> Tiramiento labial frontal</td>
</tr>
<tr>
<td> 
AU15 -> Depresión labial esquinal</td>
<td> 
AU17 -> Levantar barbilla</td>
</tr>
</table>

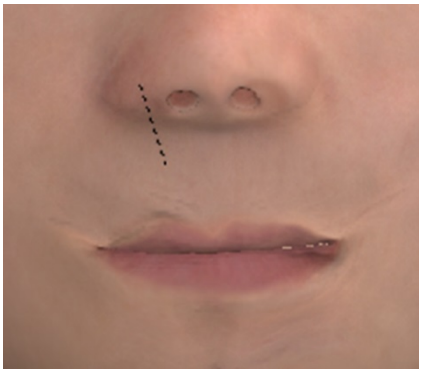
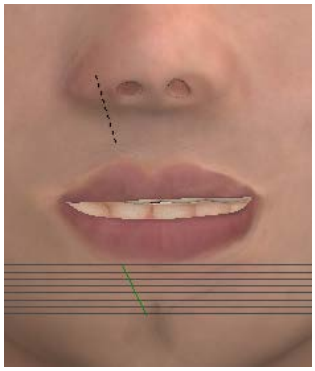
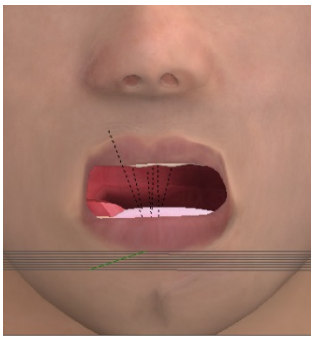
 **AU20 -> Apretar/Tensar labios**	 **AU22 -> Embudo labial**
 **AU23 -> Morder labios**	 **AU25 -> Deslizamiento labial**
 **AU26 -> Caída de la mandíbula**	

Tabla 6 - Codificación FACS de la parte inferior del rostro del avatar

Los huesos se han nombrado con el número correspondiente a la unidad de acción del FACS que realizan para diferenciarlos fácilmente (figura 3.60).

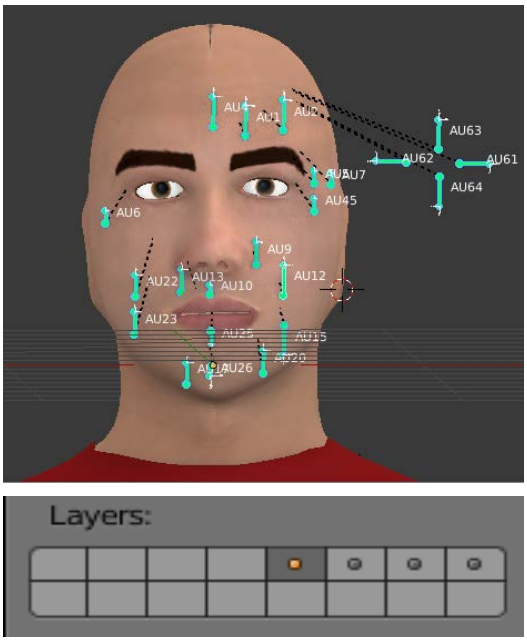


Figura 3.60 – Huesos según el FACS colocados en la capa marcada. El resto de huesos implementados previamente están en las otras capas.

Respecto a la curva F de los controladores, se ha modificado la forma de la gráfica en todas las unidades de acción para que en vez de tener un crecimiento recto, sea curvo, que es la forma ideal en la que se mueve una unidad de acción. (Apartado 2.4.3.1.2).

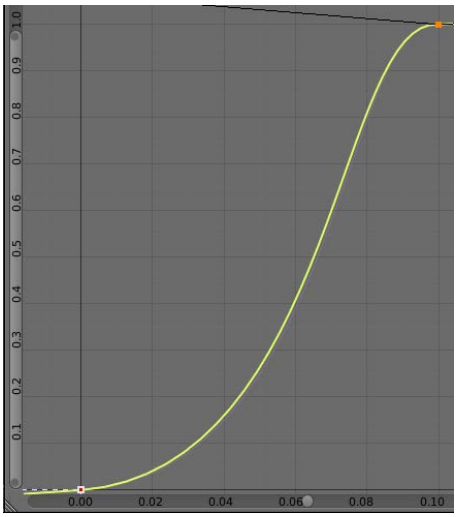


Figura 3.61 - Evolución del movimiento de las unidades de acción

El eje X de la gráfica es el movimiento del hueso de 0 cm a 10 cm en su eje local (propiedad del hueso). El eje Y de la gráfica es el valor o peso de 0 a 1, de la deformación de la forma clave (propiedad de la forma clave). Misma curva para todas las unidades de acción, para que tengan la misma evolución.

Una vez añadidas todas las unidades de acción, hay que definir el espacio que cubren en la animación. En Blender, el espacio o distancia se mide en *frames* o fotogramas. Para agregar un fotograma clave en la animación se emplea los *Keyframes*. Los *Keyframes* permiten guardar cualquiera de las propiedades de los objetos en escena (posición, rotación, escalado…) y asignarlos a un fotograma cualquiera en el editor *Action editor*, que es donde se manipula la barra de fotogramas de la animación. Los fotogramas esenciales en la unidad de acción que han de guardarse, son el estado inicial o reposo, y el estado final o de máxima intensidad. Para ello mediante la herramienta *Keyframe*, primero se ha guardado la posición y rotación inicial del hueso y se ha asignado al fotograma 0. Ahora la unidad de acción está en reposo en el fotograma 0. Posteriormente, la posición y rotación final del hueso se ha guardado y asignado al fotograma 10. De este modo, la unidad de acción está en su máxima intensidad en el fotograma 10.

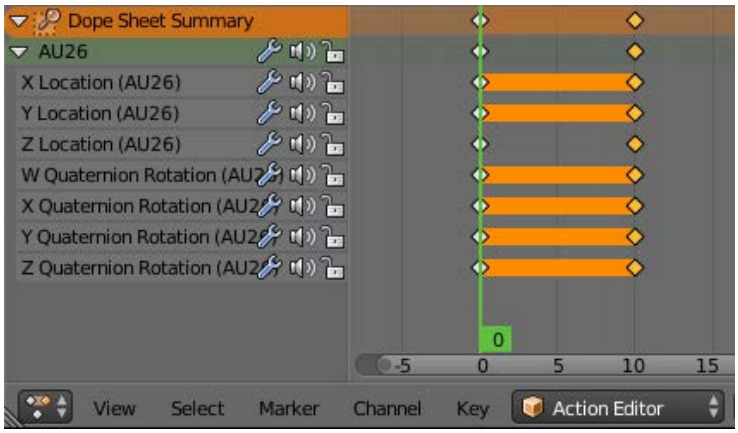


Figura 3.62 – Ejemplo de los fotogramas clave para la unidad de acción AU26

En una expresión facial, todos los micro-movimientos producidos por las unidades de acción que forman la expresión, se realizan coordinadamente. En la misma expresión nunca finaliza una unidad de acción antes que otra, o empieza una cuando termine otra. Estos movimientos no serían naturales. Por tanto para mantener dicha coordinación, en todas las unidades de acción se ha definido el estado de reposo/inicial en el fotograma 0, y el estado final/máxima intensidad en el fotograma 10. La asignación de estados puede ser en cualquier rango de fotogramas, pero en este caso se han asignado en el rango de 0-10. El espacio entre medias son los 10 fotogramas en los que la acción pasa de un estado a otro.

$$s_{UA} = 10 - 0 = 10\ fotogramas$$

La velocidad de animación se mide en fotogramas/segundos. En las propiedades de la cámara en Blender se puede editar el valor de la velocidad de animación. En un principio se ha colocado la velocidad en 12 fotogramas/segundos.

$$v_{UA} = 12\ fotogramas/segundo$$

Mediante el espacio medido en fotogramas y la velocidad de animación definida en fotograma/segundos, se puede obtener el tiempo en segundos que tarda en realizarse cada unidad de acción. El tiempo del modelo avatar en realizar cada unidad de acción en este caso es:

$$t_{UA} = \frac{s_{UA}}{v_{UA}} = \frac{10}{12} = 0.833\ \ segundos$$

La intensidad de una unidad de acción (apartado 2.4.3.1.1), se puede precisar a partir de los fotogramas. Si el estado final de la unidad de acción, que es cuando la intensidad es máxima ocurre en el fotograma 10, entonces en el fotograma 5 la intensidad de la unidad de acción será media. Siguiendo este razonamiento y los niveles de intensidad A-E que propone el FACS, se puede definir cada uno de los cinco niveles de intensidad en porcentaje (valores de 0 a 1).

Niveles de intensidad de la unidad de acción:

- A: Trazo => 0.2
- B: Leve => 0.4
- C: Pronunciado => 0.6
- D: Extremo => 0.8
- E: Máximo => 1

Multiplicando el valor del nivel de intensidad por 10, que es el número total de fotogramas, se obtiene el fotograma exacto para cada nivel de intensidad de una unidad de acción.

En el editor de juego es donde se define la lógica de la animación. Si se desea controlar cada una de las unidades de acción mediante el teclado por ejemplo, la configuración seria tal y como se muestra en la figura 3.63.

Figura 3.63 – Editor de juego. En este ejemplo la unidad AU2 se controla con la tecla "1" del teclado

- En el bloque de sensores se colocaría el sensor de tipo teclado y una tecla cualquiera. Cuando se presione la tecla se activa el controlador.
- El bloque controlador se puede poner de tipo lógico, con lógica AND, para que cuando el sensor envié la señal, el actuador realice su función.
- En el bloque actuador se colocaría la unidad de acción que se desee controlar por teclado. Se define la posición inicial y la posición final en unidad de fotogramas.

Capítulo 4 – Control de las animaciones faciales mediante ROS

4.1. Introducción

En el capítulo anterior se ha diseñado el avatar 3D por completo junto a la armadura de huesos que activa el movimiento de las formas clave del rostro. La unidad de acción realiza la función que haría un músculo, aunque no trata de ser una simulación exacta de la activación y movimiento real de los músculos faciales. Las UA solo son una herramienta del FACS para representar gestos y expresiones del rostro humano.

Una vez se ha llegado a este punto, ya se puede controlar el avatar mediante las unidades de acción por teclado para generar cualquier expresión. Sin embargo para este proyecto no resulta práctico el tener que abrir el programa, ejecutar la animación y mover manualmente las UA para generar las animaciones. Si el avatar se va a implementar como rostro para el robot, hay que buscar un método de control de las unidades de acción que en vez de ser manual, sea automático; y que su ejecución en vez de ser interna, se pueda ejecutar de forma externa, sin requerir que el programa Blender permanezca abierto en pantalla. Para resolver esta cuestión se ha decidido dividir el sistema de control del avatar en dos capas:

- **Nivel bajo de control (Blender):** Nivel encargado del control directo de los gestos del avatar mediante la implementación de las unidades de acción. Cada unidad de acción está compuesta por una forma clave y su hueso correspondiente.
- **Nivel alto de control (ROS):** En este nivel se controla qué unidades de acción actúan sobre el avatar, incluyendo el nivel de intensidad de cada unidad de acción y el tiempo de la animación. Este nivel se implementa de forma externa a Blender con ayuda del sistema operativo robótico (ROS). En el esquema de la figura 4.1, se muestran los dos niveles de control.

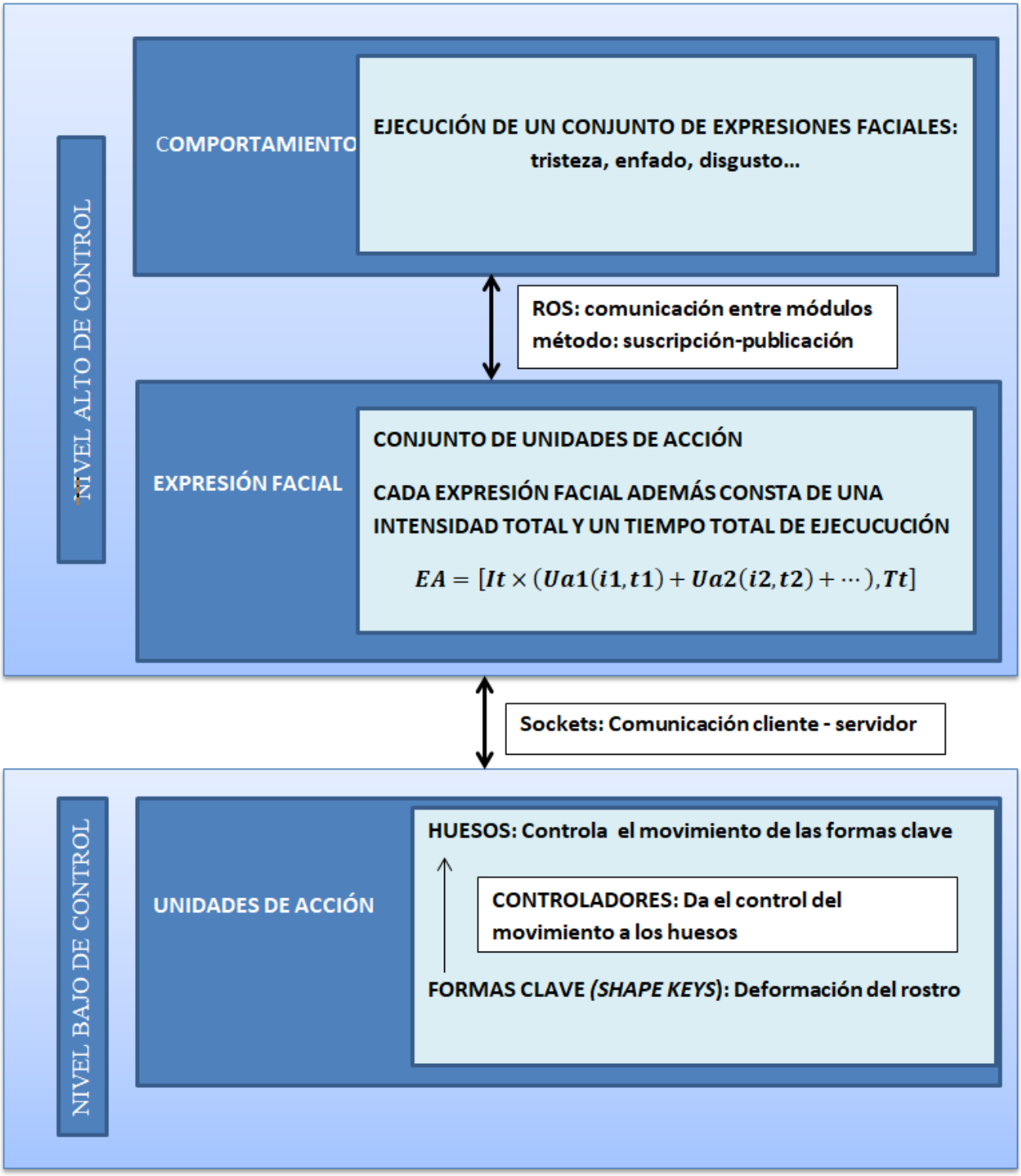


Figura 4.1 – Esquema de los niveles de control. Nivel bajo (Blender) y nivel alto (ROS)

4.2. Arquitectura y funcionamiento de ROS

Antes de entrar a explicar el nivel alto de control, se va a exponer brevemente el funcionamiento general de ROS. La información de consulta de este apartado procede de [21]. ROS (Sistema operativo robótico) realmente se trata de un meta sistema operativo, ya que ofrece las funciones de un sistema operativo, pero su instalación se realiza sobre otro sistema operativo. El que mejor soporta la instalación y uso es el SO Linux, aunque también se está adaptando a otros sistemas operativos aun de forma experimental, como: Mac OS X, Debian, Fedora y Microsoft Windows.

ROS proporciona distintos servicios desde la abstracción del hardware para el control de dispositivos, hasta la comunicación entre distintos procesos o máquinas con el fin de simplificar la tarea de crear cualquier robot. Algunas de las características que definen este sistema operativo son:

- Permite el desarrollo y uso de paquetes completos realizados por expertos en diferentes materias, que después se pueden combinar para cualquier proyecto.
- Es una plataforma multilenguaje donde se puede programar en C++, Python y en Octave, LISP y Java actualmente.
- Hace uso del protocolo TCP/IP para generar un esquema cliente-servidor. Es un sistema modular, donde cada módulo o nodo es autónomo. Los nodos tienen una dirección y cada nodo puede ofrecer o solicitar servicio de otro nodo. El servidor principal es el núcleo de ROS.

Componentes de ROS:

ROS master (maestro): Maneja la comunicación entre nodos. Cada nodo se registra al principio con el maestro. El maestro proporciona el registro de nombres y la búsqueda del resto de los nodos. Es el primer nodo que hay que arrancar, ya que si no se tiene ejecutado el maestro en el sistema, será imposible la comunicación entre nodos, servicios, y mensajes. Para arrancar el nodo maestro se debe emplear el comando: **roscore.**

Nodo: Proceso o programa ejecutable de propósito concreto. Se maneja compila y ejecuta individualmente. Están almacenados en paquetes.

Paquete: Contiene todo el código fuente de los nodos, las librerías usadas, las cabeceras y cualquier otro recurso necesario para el funcionamiento de la plataforma, la información sobre el tipo de lenguajes usados, servicios prestados, y hasta la forma de compilación.

Topics: Son etiquetas usadas como mecanismo para identificar el contenido de un mensaje y generar interacciones con los nodos. Si un nodo realiza una publicación, envía un mensaje a un *topic*. Cuando un nodo está suscrito al *topic*, recibe una llamada cuando un mensaje de este *topic* es publicado. Para que exista un *topic* debe existir por lo menos un publicador. Normalmente, siempre hay 1 publicador y n suscriptores.

Mensajes: Estructura de datos empleados en la comunicación interna entre los nodos. Los mensajes están formados por dos elementos: Un primer elemento son los campos, que indican el tipo de dato que se va a transmitir (int8, float32, string…). El segundo elemento son las constantes que dan nombre a los campos. Los mensajes están definidos en archivos .msg.

Servicio: Entregado bajo una arquitectura cliente-servidor. Se emplea un mensaje de solicitud y otro de respuesta. Un único nodo es el prestador del servicio y todos los demás pueden solicitarlo. Un nodo prestador de servicios permanece en escucha mientras espera una solicitud.

Funcionamiento de ROS

En la comunicación entre nodos, existen dos actividades principales:

- **Suscripción-publicación:** Un nodo es el encargado de publicar cierta información en un determinado canal o *topic*. Los nodos que están suscritos al *topic* reciben una notificación de que el *topic* está habilitado, y se colocan en escucha para recibir la información.

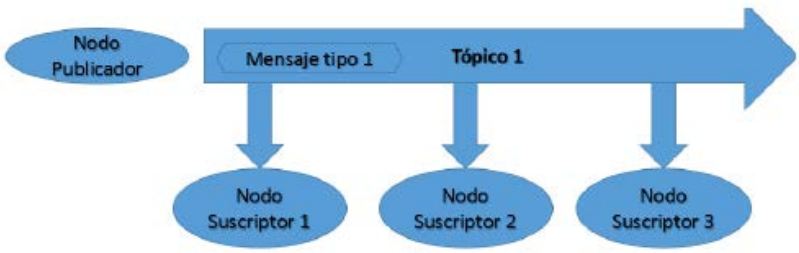


Figura 4.2 - Esquema suscripción –publicación
Imagen extraída del artículo [21]

- **Prestación-petición de un servicio:** Un nodo que emite una petición de un servicio, debe enviar unos parámetros de entrada para que el nodo que realiza la prestación, realice el servicio pedido. Una vez ejecutado dicho servicio, se genera el mensaje de respuesta, y se envía de nuevo al nodo solicitante. Si hay varios nodos que quieren

realizar la misma petición de servicios, se puede definir un orden de prioridad donde el primero que solicita, es el primero que recibe el servicio mientras se envía un mensaje de espera al resto de nodos.

Figura 4.3 - Esquema prestación – servicio
Imagen extraída del artículo [21]

Los comandos principales de uso de los nodos se muestran en la siguiente tabla.

COMANDO	FUNCIÓN
Roscore	Iniciar el nodo maestro
Rosrun nombre_paquete nombre_nodo	Ejecución de un nodo
Rosnode list	Visualizar los nodos activos
Rosnode info node_name	Visualizar información acerca de un nodo
Rostopic list	Lista activa de topics
Rostopic echo/topic	Subscribirse y copiar los contenidos de un topic
Rostopic info /topic	Ver información acerca de los topics

Tabla 7 – Comandos útiles de ROS

4.3. Instalación y configuración

En el presente trabajo la instalación de ROS se ha realizado tanto en Ubuntu como en Raspbian, una versión adaptada de Debian. Esto es porque primero se procedió a realizar el control del avatar desde un ordenador. Después se ha considerado el implementar todo el sistema, el modelo del avatar y su control, dentro de una Raspberry Pi 4, ya que es más compacto. Aun así la instalación es muy similar en ambos SO. En la página web http://wiki.ros.org/ hay un tutorial bien explicado que muestra el proceso de instalación en Ubuntu y otros sistemas operativos. En este apartado se muestra la instalación en Raspbian para la última versión hasta la fecha de ROS: la versión "ROS Melodic Morenia".

Paso 1: Instalar Dependencias y descargar los paquetes

Primero configurar los repositorios e instalar las dependencias.

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

>>    sudo apt-key adv --keyserver 'hkp: //keyserver.ubuntu.com:80' --recv-
key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654

>>    sudo apt-get update

>>    sudo apt-get install -y python-rosdep python-rosinstall-generator
python-wstool python-rosinstall build-essential  cmake
```

A continuación, inicializar rosdep y actualizarlo. Con rosdep se instalan las dependencias del sistema de un paquete.

```
>>    sudo rosdep init

>>    rosdep update
```

Después, crear un espacio de trabajo "catkin workspace", para almacenar las librerías de ROS y moverlo ha dicho directorio.

```
>>    mkdir ~/ros_catkin_ws

>>    cd ~/ros_catkin_ws
```

El modo de Instalación, seleccionarlo como: *Desktop.* Incluye el entorno de trabajo rqt, rviz y las librerías genéricas robot.

```
>>    rosinstall_generator desktop --rosdistro melodic --deps --wet-only --
tar > melodic-desktop-wet.rosinstall<br>

>>    wstool init -j8 src melodic-desktop-wet.rosinstall
```

Paso 2: Corregir problemas de instalación

Debido a que la instalación ROS en el sistema operativo Raspbian está actualmente en modo experimental, no existe un soporte oficial de ROS en este sistema operativo y pueden surgir algunos problemas durante la instalación. Para su corrección hay que proceder de la forma siguiente. En primer lugar, instalar la versión compatible de Assimp para corregir problemas de dependencia de collada_urdf**.**

```
>>    mkdir -p ~/ros_catkin_ws/external_src

>>    cd ~/ros_catkin_ws/external_src

>>    wget  http://sourceforge.net/projects/assimp/files/assimp-
3.1/assimp-3.1.1_no_test_models.zip/download -O assimp-
3.1.1_no_test_models.zip

>>    unzip assimp-3.1.1_no_test_models.zip

>>    cd assimp-3.1.1

>>    cmake .

>>    make

>>    sudo make install
```

Seguidamente, instalar el entorno de trabajo OGRE para rviz.

```
>>    sudo apt-get install  libogre-1.9-dev
```

```
>>    rosdep install --from-paths src --ignore-src --rosdistro melodic -y
```

Una vez descargados los paquetes y resueltas las dependencias hay que construir los paquetes de catkin.

```
>>    sudo ./src/catkin/bin/catkin_make_isolated --install -
DCMAKE_BUILD_TYPE=Release --install-space /opt/ros/melodic -j2
```

Ahora ROS Melodic ya estaría instalado con todos los entornos de trabajo y herramientas en una Raspberry Pi 4. Los paquetes instalados y variables se encuentran ubicados en el directorio /opt/ros/melodic. Para referirse a ellos hay que ejecutar setup.bash para indicar la dirección de ubicación y que el terminal sepa dónde están los paquetes y variables de entorno.

```
>>    echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
```

4.3.1. Directorio de trabajo

Una vez instalado ROS, todas las herramientas y paquetes se encuentran en la carpeta "ros_catkin_ws". Posteriormente, se ha decidido crear otro directorio separado para este trabajo, llamado "**catkin_ws"**. Este directorio se va a emplear específicamente para almacenar, compilar y ejecutar los ficheros de este trabajo.

Primero se crea el directorio, que en este caso va a estar ubicado en /home/pi:

```
>>    mkdir -p ~/catkin_ws/src

>>    cd ~/catkin_ws/src
```

Se ha creado el subdirectorio *src* que está actualmente vacío. De vuelta al directorio catkin_ws se compila:

```
>>    cd ~/catkin_ws/

>>    catkin_make
```

Al compilar el directorio se crean las carpetas *build* y *level*. Estas son las tres carpetas que forman el espacio de trabajo "catkin_ws".

- **Src:** Contiene los archivos de origen para los paquetes creados.
- **Build**: *cmake* se ejecuta desde aquí para construir los paquetes en src.
- **Devel:** Los archivos compilados de los paquetes se almacenan aquí para el testeo o pruebas.

Dentro de la carpeta /catkin_ws/src$ se van a guardar los archivos de mensajes en una subcarpeta llamada "avatar_msg". El modelo avatar de Blender en la subcarpeta "modelo" y en /catkin_ws/src$ se va a crear la carpeta "blender_control" donde se van a almacenar los ficheros de control del avatar.

```
>>    catkin_create_pkg blender_control rospy
```

La carpeta "blender_control" va a estar formado por tres subcarpetas:

- **"behaviour":** En esta carpeta se colocará el archivo .yaml del comportamiento que tiene que ejecutar el avatar. El usuario puede modificar el contenido de este archivo para añadir cualquiera de las expresiones que aparezca en el diccionario de expresiones. El formato .yaml es un formato de serialización de datos legible por humanos, inspirado en algunos lenguajes como C o Python.
- **"cfg":** En esta carpeta se ubicará el archivo .yaml del diccionario de expresiones. Este diccionario registra todas las expresiones que puede realizar el avatar.
- **"src":** En esta carpeta estarán los nodos o módulos de control. Los archivos de "comportamiento.py", "gestión_gestos.py" y "ros_blender_bridge.py".

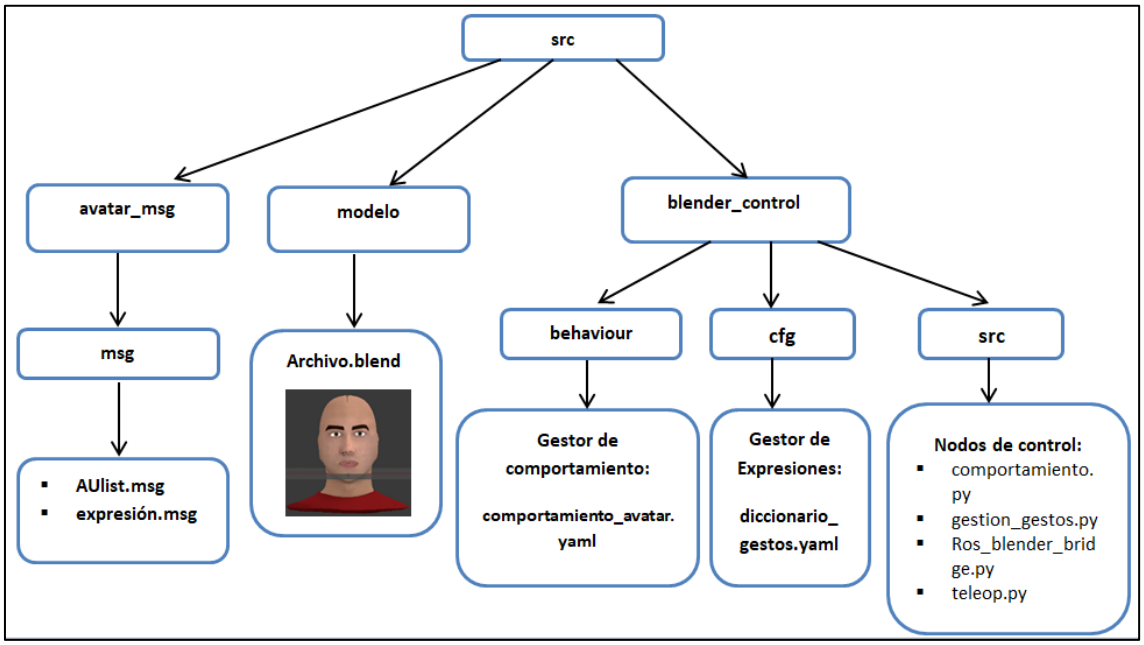


Figura 4.4 – Esquema del directorio catkin_ws/src

Es recomendable que la ubicación de las variables o directorios de la carpeta, en este caso "catkin_ws", queden guardadas cada vez que se habrá un terminal para no tener que estar direccionando la ubicación cada vez que se abre una consola nueva. Para ello hay que editar el archivo .bashrc:

```
>>    gedit  ~/.bashrc
```

Y en la última línea escribir la dirección de ubicación:

```
source /home/pi/catkin_ws/devel/setup.bash
```

Por último se guarda el archivo .bashrc, y ya quedarían guardadas las ubicaciones de las carpetas cada vez que se abre un terminal nuevo.

4.4. Nivel alto de control del avatar

El nivel bajo de control en Blender ha sido realizado en el apartado 3.4. En este apartado se va a exponer el desarrollo del nivel alto de control y su posterior ejecución. El nivel alto de control se puede dividir en dos bloques principales: un bloque de comportamiento, y un bloque de las expresiones faciales.

4.4.1. Bloque de comportamiento

Este bloque va a funcionar como un generador de comportamiento. Su función es la de recibir y procesar un conjunto de expresiones faciales entrantes, que han de ejecutarse de forma continua acorde a unos parámetros. Estos parámetros son: el nombre de la expresión, la opción de añadir UA extras a dicha expresión concreta, la intensidad total de la expresión (i_T), el tiempo de ejecución de la expresión (t_T) , y el tiempo de espera entre expresiones (t_e) .

En este bloque no se trabaja con las UA, es más el bloque no sabe que UA tienen que actuar en el avatar. Simplemente trabaja con las expresiones. Definido un comportamiento deseado, se envía cada expresión por separado con sus parámetros al bloque inferior. Posteriormente, el bloque inferior de expresiones recibirá dichas expresiones faciales que han de ejecutarse, junto a su intensidad y su tiempo de ejecución. Por último el bloque de expresiones faciales enviará a Blender las UA para activar los actuadores. A continuación, en la figura 4.5 se muestra un ejemplo de los parámetros que recibe el bloque comportamiento para su posterior procesamiento.

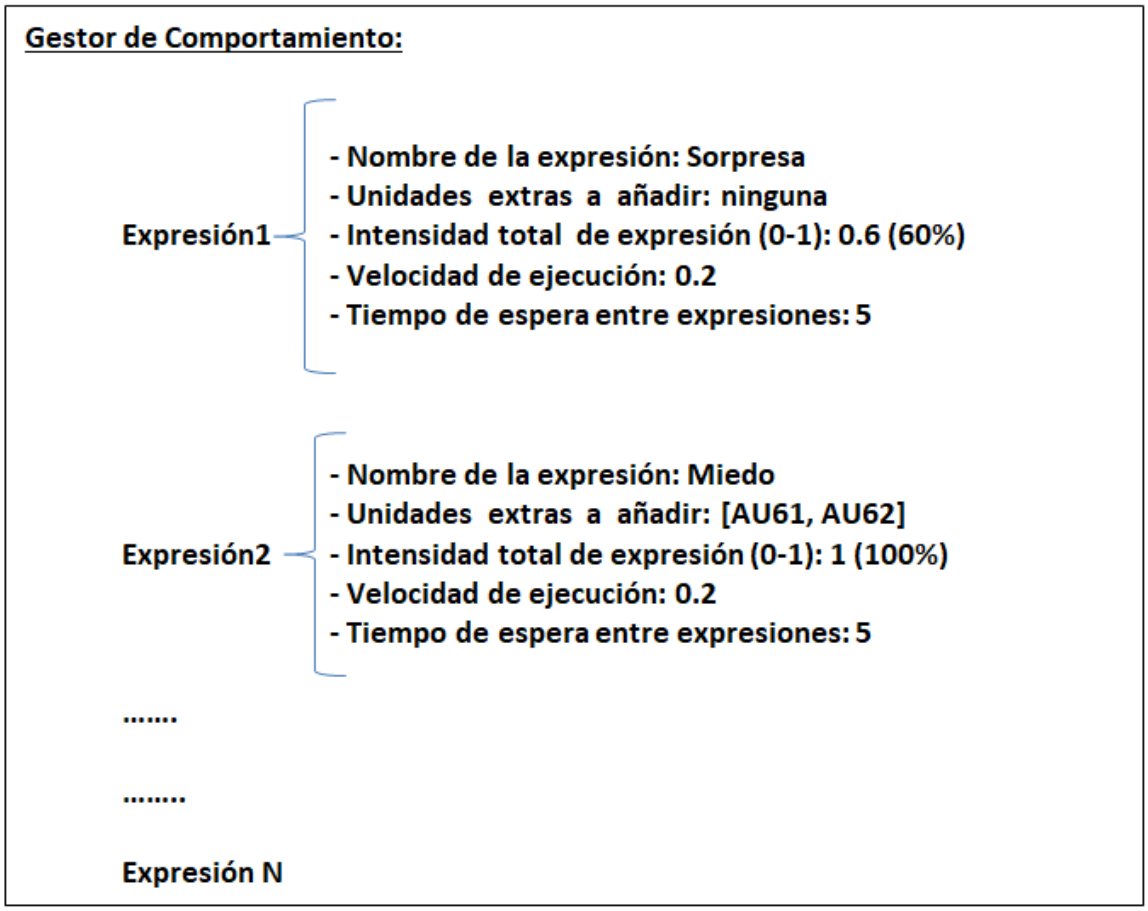


Figura 4.5 –Estructura de comportamiento_avatar.yaml. Este archivo es modificable por el usuario

4.4.2. Bloque de las expresiones faciales

Cada expresión facial está definida por un conjunto de unidades de acción (apartado 2.4.3.2). Este bloque se encarga de procesar todas las expresiones faciales que han de efectuarse procedentes del bloque superior y transformar cada expresión en un conjunto de unidades de acción. Hay que recordar que cada unidad de acción se puede expresar a partir de tres parámetros:

$$Unidad\ de\ accion\ (UA) = (UA\ ,\ i_{UA}\ ,\ t_{UA}) \quad (1)$$

UA es el número de la acción según el FACS. La intensidad de la UA (i_{UA}) tendrá un valor de 0-1 (apartado 3.4.3). Y el tiempo de ejecución de la UA (t_{UA}), se calcula como se indica en el apartado 3.4.3. Este valor va a ser no modificable, salvo que se cambien los *frames* de cada acción o la velocidad ($frames \times segundo$) dentro de Blender. Es por eso que al trabajar con las unidades de acción en ROS, estas se van a representar solo con los dos parámetros siguientes:

$$Unidad\ de\ accion\ (UA) = (UA\ ,\ i_{UA}) \quad (2)$$

En ROS la expresión facial se va a expresar únicamente como una extensión del vector de las unidades de acción. Ejemplo: Expresión cualquiera formada por UA1, UA2 y UA4 con sus intensidades máximas. El vector correspondiente es:

$$UExpresión\ Facial = [1,\ 5,\ 2,\ 5,\ 4,\ 5] \quad (3)$$

Además de las unidades de acción, cada expresión facial también va a estar definida por su tiempo de ejecución (t_T) y su nivel de intensidad total (i_T), ambos parámetros también recibidos del bloque superior. El tiempo de ejecución es el tiempo empleado en realizar la expresión al completo desde que todas las UA están en reposo, hasta que alcanzan su correspondiente posición final. El nivel de intensidad total de la expresión (i_T) es diferente el nivel de intensidad de cada unidad de acción (i_{ua}). La i_{UA} indica la magnitud del movimiento de cada UA. La i_T mide en porcentaje, el grado de intensidad de toda la expresión facial. Este porcentaje va a afectar a todas las UA que componen la expresión por igual. La i_T tiene un rango de valores de 0-1 (0-100%).

Por lo tanto cada expresión se puede definir a partir de su tiempo de ejecución y su magnitud, formada como la suma total de las UA que la componen (cada una con su intensidad y tiempo concreto) y multiplicado por la intensidad total de la expresión. La fórmula teórica para la expresión facial sería:

$$Expresión\ Facial = [i_T \times (UA_1(i_1, t_1) + UA_2(i_2, t_2) + \cdots), t_T] \quad (4)$$

Los datos de salida de este bloque se calcularán a partir de dos factores:

- De la consulta de un "diccionario de expresiones" donde vendrán listadas todas las expresiones que es capaz de realizar el avatar virtual. Y donde cada expresión vendrá definida por un vector de todas las UA que actúan y la intensidad de cada una (i_{ua}). Este diccionario de expresiones se puede modificar por el usuario.
- De las expresiones entrantes procedentes del bloque superior de comportamiento, que son las expresiones que realmente tiene que ejecutar el avatar.

Sabiendo que expresiones tienen que ejecutarse (comportamiento) y que UA conforman cada una de todas las expresiones existentes en el avatar (diccionario), se puede obtener que UA son estrictamente necesarias para su envío al actuador de Blender. En el siguiente apartado se muestra la comunicación modular, entre los bloques y los dos niveles de control.

4.5. Comunicación entre los módulos y los dos niveles de control

Módulo: comportamiento.py

El primer paso se lleva a cabo en el módulo "comportamiento.py". Este módulo carga el archivo de entrada "comportamiento_avatar.yaml" en el que estará definido el comportamiento del avatar. Este archivo .yaml tiene la forma de la figura 4.5.

"Conportamiento.py" leerá el archivo de entrada, y separará los datos en cinco categorías: exp (expresión), el vector au_ext (con UA extras que se deseen añadir), i_T , t_T y t_e . Para definir estas variables, el módulo ha abierto el mensaje "expresion.msg" dentro de la carpeta "avatar_msg.msg", para saber el tipo de datos y el nombre de los datos con los que trabajar.

```
string exp
float32[] au_ext
float32 it
float32 tt
```

Figura 4.6 – Mensaje expresión.msg donde se muestra el tipo de datos y nombre

Por último, "comportamiento.py" realizará una publicación donde estarán presentes estos datos. Esta publicación se llama "*topic_expresion*". Este módulo no mostrará nada por pantalla, solo realiza la publicación para que los nodos que se suscriban a ella, visualicen la información.

Módulo: gestion_gestos.py

El siguiente paso lo realiza el módulo "gestión_gestos.py". Este módulo se suscribirá a "*topic_expresion*", por lo que tendrá acceso a los datos publicados: exp, au_ext, i_T , t_T . A continuación, "gestión_gestos.py" abrirá el diccionario de gestos para consultar todas las expresiones que puede realizar el avatar.

```
# ARCHIVO DE CONFIGURACION DE GESTOS.

# forma del vector de unidades de acion
# aus: [unidad_de_accion_1, intensidad_1, unidad_de_accion_2, intensidad_2 ...]

neutra:
    aus: [1, 0, 2, 0, 4, 0, 5, 0, 6, 0, 7, 0, 9, 0, 10, 0, 12, 0, 13, 0, 15, 0, 20, 0, 22, 0, 23, 0, 25, 0, 26, 0, 61, 0, 62, 0, 63, 0, 64, 0]

alegria:
    aus: [6, 1, 12, 1]

tristeza:
    aus: [1, 1, 4, 1, 15, 1]

sorpresa:
    aus: [1, 1, 2, 1, 5, 1, 26, 1]

ira:
    aus: [4, 1, 5, 1, 7, 1, 23 ,1]

asco:
    aus: [9, 1, 20, 1, 25, 1]

desprecio:
    aus: [4, 1, 9, 1, 10, 1, 45, 1]

miedo:
    aus: [1, 1, 2, 1, 4, 1, 5, 1, 7, 1, 20, 1, 26, 1]

enfado:
    aus: [4, 1, 7, 1, 17, 1]
```

Figura 4.7 –Archivo diccionario_gestos.yaml. Cada gesto está formado por un vector de unidades de acción e intensidad

Después de esta consulta y con los datos obtenidos de "comportamiento.py", el módulo realizará una publicación llamada "topic_au". En esta publicación expondrá los datos de la i_T , t_T y el vector de las unidades de acción (au). La definición de estas variables proviene del mensaje "AUlist.msg".

```
float32 it
float32 tt
float32[] au
```

Figura 4.8- Mensaje AUlist.msg donde se muestra el tipo de datos y nombre

Al ejecutar "gestión_gestos.py" se imprimirá por pantalla, un vector de las unidades de acción au[], que están actuando. X e Y representan dos unidades de acción distintas:

$$[X, i_X , Y, i_Y , \dots] \quad (5)$$

Ros_blender_bridge.py

Este módulo sirve de puente entre los dos niveles de control, entre ROS y Blender. Para establecer la comunicación entre ambos niveles se emplean sockets. Es por eso que el módulo "ros_blender_bridge.py" funcionará a la vez como un nodo de ROS para establecer contacto con "gestión_gestos.py", y también hará la función de cliente en la comunicación con Blender mediante sockets. El presente módulo se va a suscribir a la publicación "topic_au" por lo que estará en disposición de obtener los datos publicados por "gestión_gestos.py". La función de este módulo es transformar el vector recibido de la expresión facial en la matriz siguiente:

$$[X, i_X, Y, i_Y, \dots] \tag{5}$$

$$[(X, pos\_inicialX_{expr}, pos\_finalX_{expr}, t_T), (Y, pos\_inicialY_{expr}, pos\_finalY_{expr}, t_T), \dots] \tag{6}$$

que son los parámetros que procesa el actuador de Blender. La posición inicial y final de cada UA hacen referencia a la posición inicial y final de cada uno de los huesos que controla el movimiento de las formas clave.

$pos\_inicialX_{expr}$ = Es la posición o fotograma inicial de la unidad de acción X, en la expresión actual. Se ha inicializado el valor a 1 para el primer ciclo o expresión. Después este valor se va a ir actualizando en cada UA cada vez que se realiza una nueva expresión. El valor actualizado es $pos\_inicialX_{expr} = pos\_inicialX_{expr-1}$. Ya que cada UA de la expresión actual debe comenzar desde la posición final de esa misma UA en la expresión anterior, para que el movimiento del avatar sea natural.

pos_finalX_{expr}= Es la posición o fotograma final de la unidad de acción X en la expresión actual. Este valor se calcula como el producto de la intensidad de la unidad de acción X, por el valor 10 y por la intensidad total de la expresión actual.

$$pos\_finalX_{expr} = i_T \times i_X \times 10 \tag{7}$$

Mediante sockets se enviará la matriz de datos anterior (ecuación 6), al modelo de Blender. La ip utilizada es "127.0.0.1" y el puerto 52248.

Módulo: ua_avatar_blender.py

Archivo Python cargado en el modelo del avatar, que recibe la matriz de datos procedentes de "ros_blender_bridge.py". Este módulo crea el socket y hace de servidor en la comunicación con "ros_blender_bridge.py". Una vez obtenida la matriz, este módulo se encarga de separar los elementos de la matriz y asignar cada elemento, a las características del actuador de Blender, situado en el editor lógico.

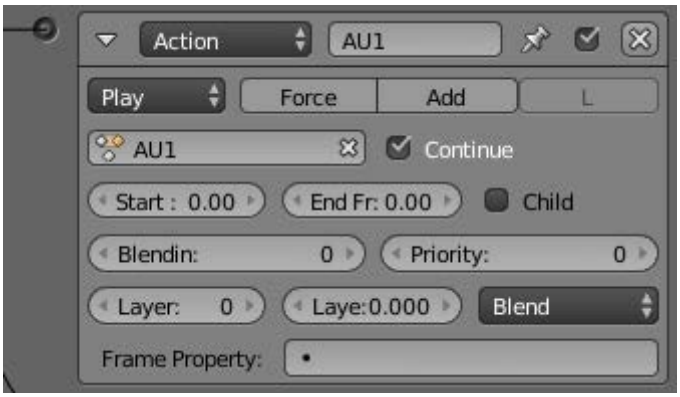


Figura 4.9 –Parámetros del actuador del editor lógico del Blender. Este bloque es para la AU1

En el editor lógico, no se asigna ningún valor a los parámetros de los actuadores. Se dejaran en cero, ya que "ua_avatar_blender.py" se ocupa de asociar cada elemento de la matriz a los parámetros del actuador.

```
for i in mensaje:
    name = "AU"+str(int(i[0]))
    start = int(i[1])
    end = int(i[2])
    priority = 1
    layer = layer
    blendin = 1
    mode = 0
    layerWeight = 0
    ipoFlags = 1
    speed = i[3]
    au = action_unit(name, start, end, priority, layer, blendin, mode, layerWeight, ipoFlags, speed)
    grupo_au.append(au)
    layer += 1
```

Figura 4.10 – Datos recibidos de ros_blender_bridge.py para asignarlos a las características del actuador de Blender

$int(i[0]) = X(unidad\ de\ accion\ número\ X\ , AUX)$
$int(i[1]) = \ pos\_inicialX_{expr}$
$int(i[2]) = pos\_finalX_{expr}$
$int(i[3]) = \ t_T$

El resto de parámetros (*priority, layer, blendin, mode, layerWeight, ipoFlags*) se dejarán con los valores predeterminados de actuación. La configuración global del editor lógico se observa en la figura 4.11.

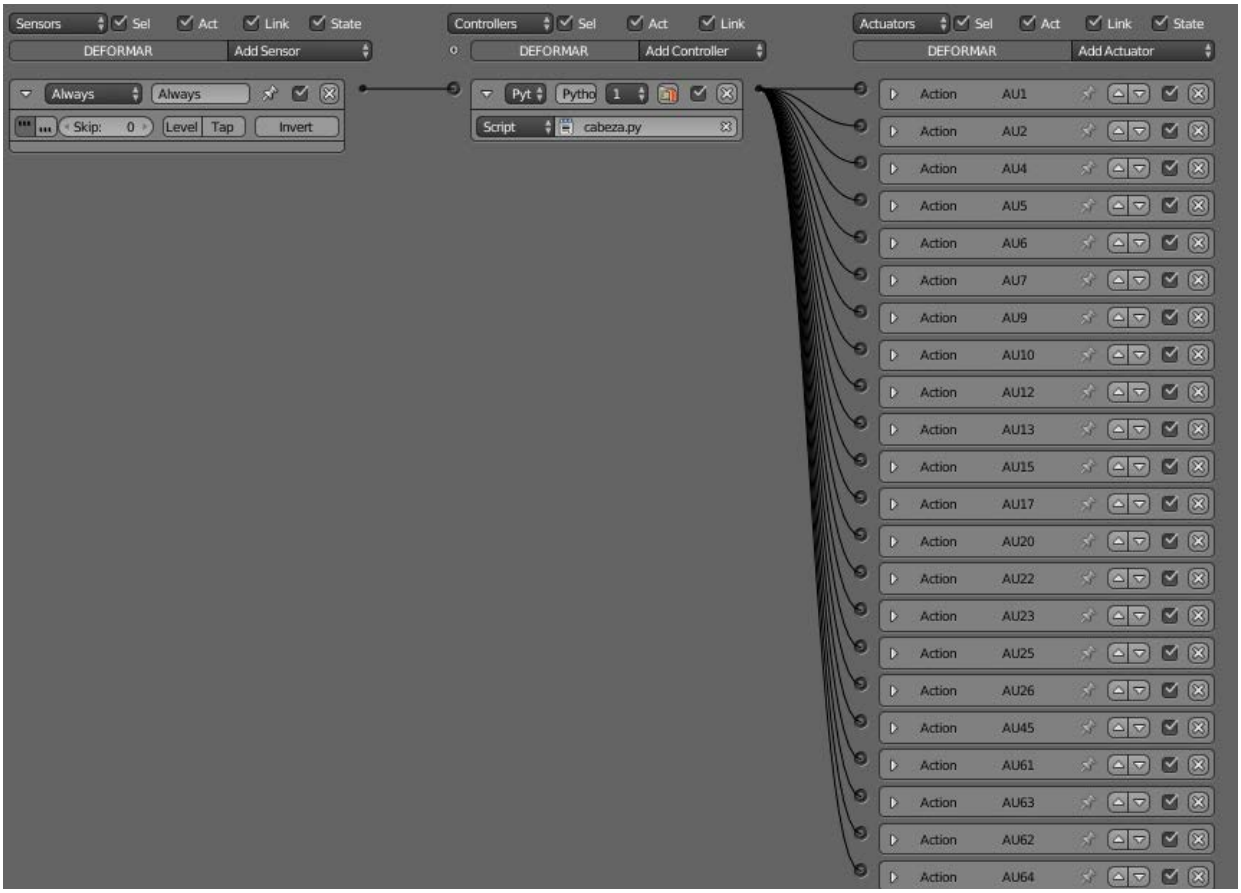


Figura 4.11 – Configuración del editor lógico con todas las unidades de acción implementadas en el avatar

A diferencia de la animación controlada manualmente por teclado (figura 3.63), el bloque del sensor ahora se deja en un valor de *always*. Esto indica que el sistema estará a la espera de que el controlador actúe por sí solo. Ahora el controlador no se activará mediante ningún sensor. El controlador será de tipo *script*, donde se colocará el archivo "ua_avatar_blender.py". En los bloques actuadores se colocarán todas las unidad de acción del avatar con los parámetros tal y como se muestra en la figura 4.9 y conectadas al controlador.

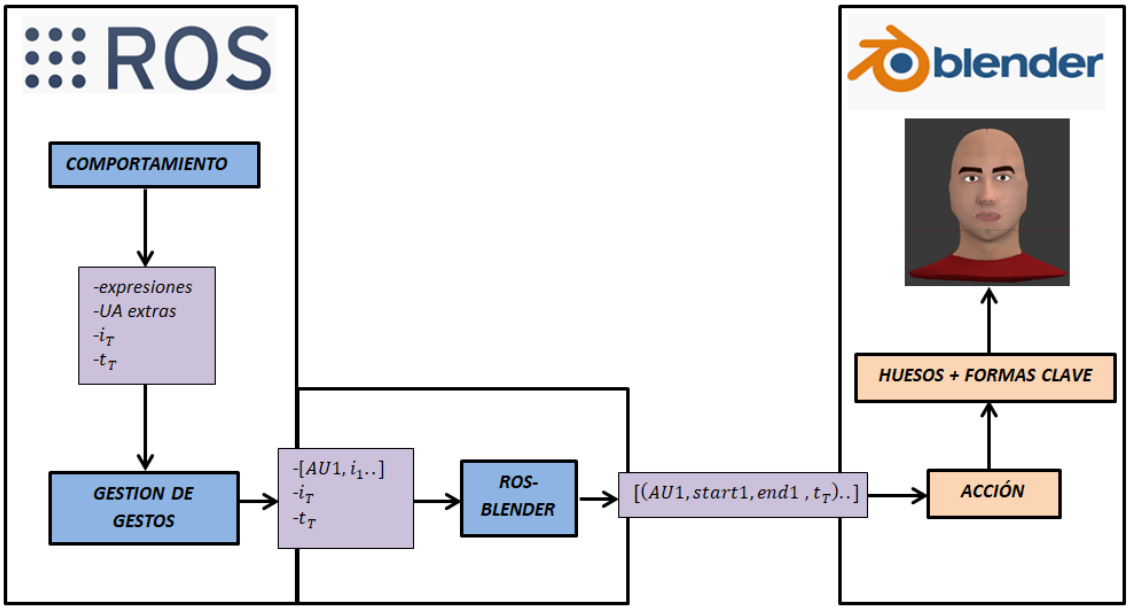


Figura 4.12 – Esquema resumen del control del avatar por medio de las unidades de acción

Capítulo 5 – Voz y sincronización labial en tiempo real

5.1. Introducción

Además de los movimientos gestuales y expresiones del rostro, otra de las funciones complementarias que puede tener al avatar es un sistema de voz y sincronización labial en tiempo real. La sincronización labial o fonomímica consiste en la sincronización de los movimientos labiales mediante el uso de visemas, con la parte sonora de la comunicación verbal que es el habla. Los visemas son la representación visual de los fonemas del lenguaje oral, a través de las expresiones labiofaciales. Como aplicación práctica, este sistema de sincronía con el habla, puede permitir al avatar establecer comunicación verbal con un receptor en tiempo real.

Como punto inicial, primero se ha decidido realizar una búsqueda previa de posibles sistemas de sincronización labial en tiempo real, en avatares virtuales. El resultado no ha sido exitoso. Se ha encontrado algún caso, como la aplicación "*SALSA Lip Sync*" pero esta es una aplicación de pago y es específica para la plataforma Unity. En el entorno Blender no existe aplicación para este cometido. Sí que existen algunos programas como "*Papagayo Lipsync*" o "*Rhubarb Lipsync*" que se pueden instalar como extensiones dentro de Blender, pero estos no realizan una sincronía labial del modelo en tiempo real. Estos programas realizan la sincronía labial de un archivo de audio/voz pregrabado y cargado previamente en Blender, en la tabla de tiempos. Así, el programa de sincronía ya puede conocer los tiempos y asociar cada visema diseñado en Blender, con su fonema correspondiente, en el instante exacto del archivo de audio. Estos programas pueden ser útiles para la realización de películas o cortos de animación, donde no hay eventos que ocurran en tiempo real de ejecución, ya que la animación está grabada previamente. Sin embargo, para el caso presente no es de utilidad ya que la sincronía ha de ser en tiempo real. En el apartado siguiente se mostrará una solución y el método empleado.

5.2. Metodología

La metodología a seguir se muestra en la figura 5.1. La entrada de datos es un texto que el emisor escribe por pantalla. Para el habla se va a emplear un convertidor texto a voz *text2speech.* De forma paralela, se va a crear un algoritmo que divida el texto de entrada en un vector de fonemas. Posteriormente, cada uno de estos fonemas se va a asignar a su correspondiente visema creado en Blender. Por último, tanto el sonido generado, como la generación de visemas se ejecutarán de forma simultánea en la animación.

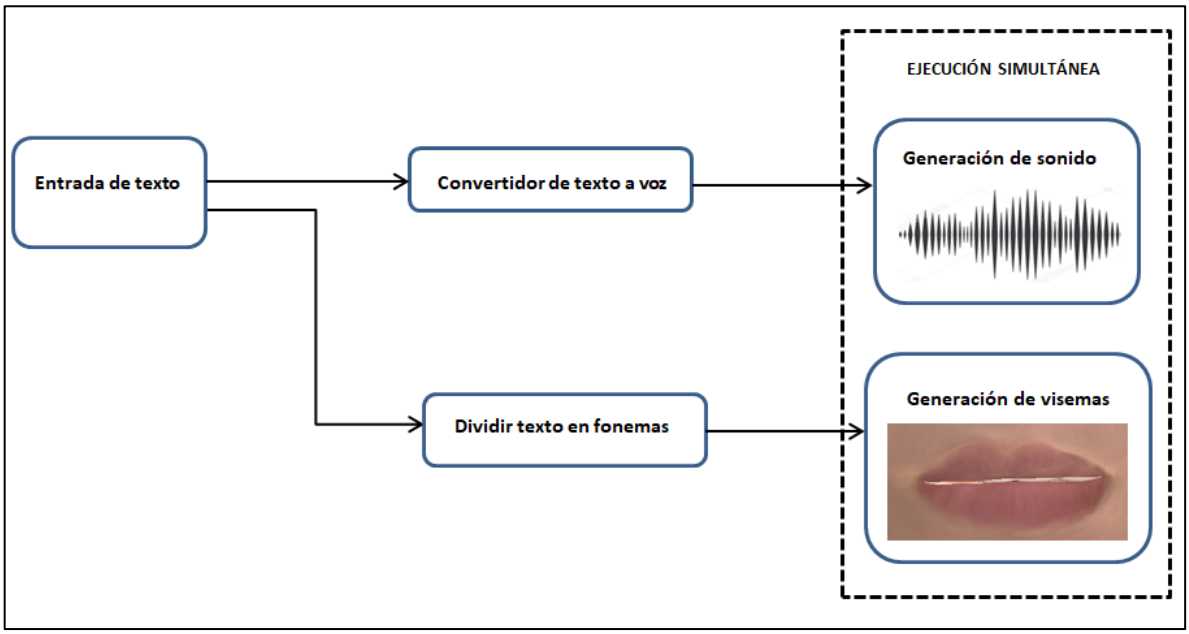


Figura 5.1 – Metodología del sistema empleado

El sistema en tiempo real requiere de una ejecución automática de los movimientos, sin necesidad de activar manualmente cada actuador en un fotograma dado dentro de Blender. Por esta razón, se necesita algún módulo que se conecte con Blender de forma externa y le envíe qué unidad de acción (visema) tiene que actuar en cada instante. Para ello, se van a emplear dos módulos conectados mediante sockets.

- Módulo externo, cliente: Transforma el texto entrante en un vector de fonemas. De forma simultánea, transforma el texto en voz y envía secuencialmente los fonemas al segundo módulo ubicado dentro de Blender. (ver apartado 5.4).
- Módulo de Blender, servidor: Recibe secuencialmente cada uno de los fonemas del vector de fonemas. Este módulo asigna cada fonema a su visema creado en Blender. Por último, se ejecuta el actuador de cada visema de forma secuencial, creando así la sincronización labial del habla en tiempo real. (ver apartado 5.5)

5.3. Creación de visemas

El primer paso es crear los visemas que empleará el modelo avatar durante el habla. Cada visema representará a su correspondiente fonema del lenguaje oral. El proceso es el mismo que cuando se crearon las unidades de acción del rostro en Blender. Empleando el sistema de huesos del avatar, se crea el gesto labial de cada visema. Para facilitar su designación, cada visema se va a etiquetar del mismo modo que las unidades de acción con el formato "AU + número".

En el idioma español existen un total de 22 fonemas, pero para este trabajo se van a representar los visemas asociados a los fonemas más utilizados. Estos fonemas serán los fonemas vocálicos, y algunos fonemas consonánticos. En la tabla 8 se muestra los visemas creados para el avatar.

Fonema /a/	Fonema /e/
	
Fonema /i/	**Fonema /o/**
	
Fonema /u/	**Fonema /s/**
	
Fonema /f/	**Fonemas /b/-/m/-/p/**
	

Tabla 8 – Tabla de los visemas asociados a cada fonema

5.4. Módulo externo

El módulo externo es el archivo Python llamado “lipsync_external.py”. Este módulo funciona como cliente y envía los datos al módulo de Blender. En este apartado se expone el funcionamiento del módulo.

Por un lado se va a implementar el convertidor que transforme el texto escrito por el usuario en la consola, en habla. Existen varios convertidores *text2speech* aunque dentro de los gratuitos, los más completos son *gTTS* y *pyttsx3*, cada uno con sus ventajas e inconvenientes.

Google Text-to-Speech (gTTS)

Librería Python que permite conectar con la API de Google, convertidor de texto a voz. Este es un convertidor muy eficiente que permite la lectura de longitudes de texto manteniendo la entonación, pronunciación, abreviaturas etc. La voz es personalizable en varios idiomas incluido el español. La velocidad del habla también se puede modificar, aunque solo en dos posiciones (rápido y lento). La gran ventaja de *gTTS* respecto a la mayoría de convertidores actuales, es que la voz generada suena realista y natural. Hay que recalcar que de momento solo se ha implementado voz femenina, lo cual en este caso es una desventaja ya que el avatar es masculino. También requiere de conexión a internet para conectarse con Google.

Además de *gTTS* también hay que instalar varias librerías para que Python pueda interpretar el reconocimiento de voz y ejecutar el archivo de audio transformado. A continuación se muestran los comandos de instalación:

```
$ pip install gTTS

$ pip install SpeechRecognition

$ pip install playsound

$ pip install pyAudio
```

Una vez finalizada la instalación de las librerías, en el presente módulo "lipsync_external.py" se implementa la función "speak2". Esta función recibe como argumento el texto escrito por el usuario. Se configura el idioma y velocidad de voz y a continuación el convertidor *gTTs* transforma el texto en habla. Después guarda el archivo de audio generado con el nombre "voice.mp3". Por último se reproduce dicho archivo.

```
#funcion que transforma texto a voz utilizando el convertidor GoogleText2Speech
def speak2(text):
    tts = gTTS(text=text, lang="es",slow=True)
    filename= "voice.mp3"
    tts.save(filename)
    playsound.playsound(filename)
```

Figura 5.2- Función que ejecuta el convertidor gTTS

Convertidor Text-to-Speech (pyttsx3)

Esta es otra librería de Python que permite la conversión de texto a voz. A diferencia de otras librerías alternativas, no requiere conexión a Internet y es compatible para Python 2 y Python 3. Este convertidor también permite personalizar el idioma dentro de un gran catálogo. También permite modificar los parámetros de volumen y la velocidad del habla en una escala de valores de 0 a 200. A diferencia de *gTTS* que llevaba implementada en la aplicación su voz natural propia, *pyttsx3* tiene soporte para el uso de varios sintetizadores de voz. Esto hace que la voz en *pyttsx3* suene más robótica y menos realista que en el caso del convertidor de Google *gTTS*. Sin embargo en este convertidor sí que se puede personalizar tanto voz masculina como voz femenina, lo cual es una ventaja.

El comando de instalación es el siguiente:

```
$ pip install pyttsx3
```

En este mismo módulo se ha añadido también la función “speak1”. Esta función recibe como argumento el texto escrito por el usuario. El sintetizador de voz empleado es *eSpeak*. Como parámetros a personalizar, se ha configurado el idioma a español y a continuación se ha establecido un valor medio de velocidad de habla. Por último el convertidor llama a la función “say” que transforma el texto en voz, y se reproduce la voz.

```
#funcion que transforma texto a voz utilizando el convertidor pyttsx3
def speak1(text):
        engine = pyttsx3.init(driverName='espeak')
        voices = engine.getProperty('voices')
        engine.setProperty('voice', voices[19].id) #voz 19 es el idioma espanol
        rate=engine.getProperty('rate')
        engine.setProperty('rate', 125) #velocidad de habla

        engine.say(text)
        engine.runAndWait()
        engine.stop()
```

Figura 5.3- Función que ejecuta el convertidor pyttsx3

Aunque en la ejecución solo se vaya a emplear uno de los convertidores, se ha decidido añadir en el módulo “lipsync_external.py” ambas funciones por si se desea emplear un sistema de conversión u otro.

Extracción y envío de fonemas

El otro proceso que hay que llevar a cabo es discriminar los fonemas o letras que representan los visemas diseñados, dentro del texto de entrada. El primer paso es dividir el texto de entrada en un vector de palabras, llamado "vect_palabras[i]". Dentro de este vector, cada elemento i (palabra) está compuesto a su vez por un vector de letras, "vect_letras[j]". Recorriendo ambos bucles, al final se terminan evaluando todas las letras del texto. Cada vez que una letra coincida con uno de los fonemas de la tabla 8, se introduce el elemento en un nuevo vector, "vect_fonemas[z]" que inicialmente está vacío. Los fonemas se van colocando ordenadamente para después enviarlos secuencialmente a Blender. Así después, los visemas se representarán también conforme al orden del texto de entrada. Para el envío de fonemas se ha creado la función "enviar_fonemas". Esta función coge el vector "vect_fonemas[]" y va enviando uno por uno en orden de entrada, los elementos de este vector al módulo de Blender cada 0.3 segundos (figura 5.4).

Ejecución concurrente

Por lo general, las instrucciones de un código siguen una ejecución secuencial ya que se van ejecutando una tras otra en el orden en el que están escritas. Sin embargo existen situaciones donde se necesita que dos funciones se ejecuten de forma concurrente o en paralelo. Este es el caso, ya que se necesita que el habla y la reproducción de visemas se reproduzcan simultáneamente para conseguir esa sincronía. Para ello se puede emplear la librería *multiprocessing.* Este es un paquete que permite llevar a cabo varias operaciones de un programa de modo concurrente. En la figura 5.4 se muestran las dos funciones que han de actuar en paralelo. Por un lado la función "hablar" y por otro lado la función "enviar_fonemas". La función "hablar" ejecuta uno de los dos convertidores texto a voz expresados previamente. La función "enviar_fonemas" envía los datos del vector de fonemas al módulo de Blender para que este instantáneamente reproduzca los visemas asociados a los fonemas que recibe. Para asignar la concurrencia de las funciones se emplea *multiprocessing.Process*. Una vez que se han ejecutado ambas funciones, el vector de fonemas se vacía para la siguiente frase de entrada.

```
      def hablar():
       s1=speak1(frase) #convertidor pytssx3
       #s=speak2(frase) #convertidor gtts

      def enviar_fonemas():
       time.sleep(0.2)
       for z in range(len(vect_fonemas)):
         clientesocket.send(vect_fonemas[z])
         time.sleep(0.3)
         z=z+1

      #ejecutar ambas funciones en paralelo:
      p1 = multiprocessing.Process(target=hablar)
      p2 = multiprocessing.Process(target=enviar_fonemas)
      p1.start()
      p2.start()
      p1.join()
      p2.join()

      vect_fonemas=[] #vaciar el vector para la siguiente frase a escribir

    clientesocket.close()
return sock
```

Figura 5.4 – Ejecución paralela de las dos funciones

5.5. Módulo Blender

El módulo interno es el archivo Python llamado “lipsync_blender.py”. Este módulo se carga dentro de Blender y funciona como servidor en la comunicación. Su función es ir recibiendo los datos que le llegan y activar los actuadores correspondientes para representar los visemas en la animación.

En la tabla 9 se muestra el código del actuador del visema para cada fonema.

Fonemas	Código de actuador del visema
“a”	AU90
“e”	AU91
“i”	AU92
“o”	AU93
“u”	AU94
“s”	AU95
“f”	AU96
“b-m-p”	AU97

Tabla 9 – Código de los actuadores de los visemas

En el editor lógico, no se asigna ningún valor a los parámetros de los actuadores. Se dejarán en cero (figura 5.5) al igual que se hizo con los bloques actuadores de las unidades de acción de las expresiones faciales. Estos parámetros se van a configurar dentro del archivo "lipsync_blender.py", ver figura 5.6.

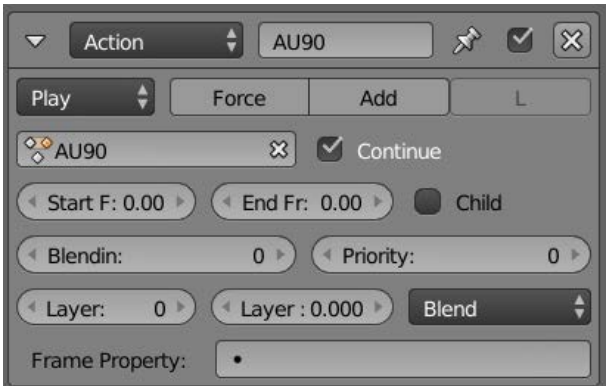


Figura 5.5- Representación de un bloque actuador, en este caso el actuador AU90

```
#parametros de los bloques actuadores
name = "AU" + str(int(viseme))  #Nombre del visema como unidad de accion: AU90,AU91,AU92...
print(name)
start = 0
end = start+50
priority = 1
layer = 1
blendin = 0
mode = 0
layerWeight = 0
ipoFlags = 0
speed = 10

#crear objeto "au_viseme" para la clase action_unit con los parametros anteriores
au_viseme = action_unit(name, start, end, priority, layer, blendin, mode, layerWeight, ipoFlags, speed)

#ejecutar la funcion "activate" para el objeto creado. Con "activate" se realiza la accion
au_viseme.activate(own)
```

Figura 5.6 – Valores de los parámetros de los bloques actuadores en el archivo lipsync_blender.py

Cuando el módulo reciba un dato, se le dará un valor (90-97) según el fonema que sea (ver tabla 9). Este valor se almacena en la variable *viseme*. El parámetro *name* permitirá diferenciar el bloque actuador de cada visema. El valor del resto de parámetros del actuador (fotograma inicial y final, prioridad, capa, velocidad…) será el mismo para todos los bloques. Por último la función *activate*, llama a la función *playAction*. Esta función pone en marcha el actuador y realiza el visema en la animación. La configuración del editor lógico se muestra en la figura 5.7.

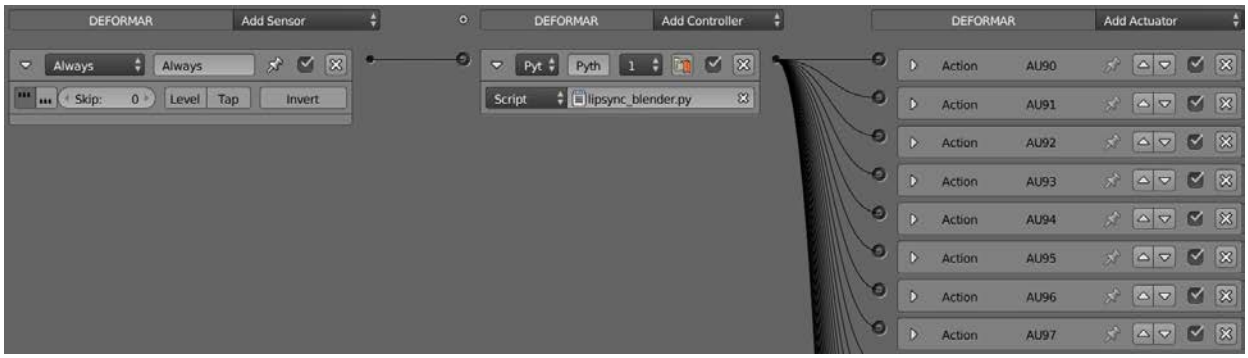


Figura 5.7- Configuración del editor lógico con el controlador y los actuadores empleados

Como se muestra en el editor lógico, el bloque del sensor se deja en un valor de *always.* Esto indica que el sistema estará a la espera de que el controlador actúe por sí solo. El controlador no se activará mediante ningún sensor. El controlador será de tipo *script*, donde se colocará el módulo "lipsync_blender.py". En los bloques actuadores se colocarán todos las visemas con los parámetros tal y como se muestra en la figura 5.5 y conectados al controlador que es el archivo Python.

Capítulo 6 – Implementación del sistema avatar-máscara

6.1. Introducción

En el presente capítulo se va a exponer la forma de implementar el avatar en el robot. Después del análisis de otros proyectos relacionados con la retroproyección de rostros en cabezas robóticas (apartado 2.3), se ha decidido crear una máscara traslucida para proyectar la imagen del avatar y que forme el rostro del robot. El capítulo está dividido en dos secciones. Una primera sección dedicada al diseño y fabricación de la máscara, apartados 6.2 y 6.3. Y otra sección dedicada a la retroproyección de la imagen del avatar sobre la máscara creada, apartado 6.4. En esta sección se muestra la elección del proyector y el sistema de proyección empleado para que la imagen quede correctamente ajustada a la forma de la máscara.

6.2. Diseño de la máscara

En este apartado se va a mostrar los distintos pasos para el diseño de la máscara. El diseño de la máscara se ha realizado con el programa Blender, al igual que se ha hecho con el avatar. De este modo la edición de la forma y dimensiones del objeto resulta más sencilla al ser un formato de malla donde se pueden modificar, añadir o eliminar vértices. Una vez que la figura de la máscara esté realizada, se ajustará el tamaño y forma de la máscara a la ranura de unión casco-máscara. Cuando la máscara esté completamente diseñada en Blender, se transformará el archivo.blend a archivo.stl. En el trabajo Fin de Grado realizado de forma paralela al actual, este archivo .stl de la máscara se unirá con el casco, diseñado en el programa CATIA. Por último, cuando las medidas estén correctas, tanto la máscara como el casco, por partes, se fabricarán en una impresora 3D.

Para facilitar el trabajo y no empezar de cero a construir la máscara, se ha tomado como punto de partida, el avatar realizado en Blender. A partir de la cabeza del avatar se ha ido moldeando la máscara. Esta mascara presentará la geometría de un rostro pero no tendrá rasgos faciales, ya que va a servir para proyectar el avatar virtual sobre ella. Por esta razón, salvo la zona de la nariz, que mantendrá su forma, el resto de la máscara será de geometría suave.

El avatar estaba compuesto por la cabeza completa. Por su parte, la máscara en vertical, va a tener que abarcar desde el comienzo del cuello hasta la parte superior de la frente y en distancia horizontal, toda la parte de los pómulos hasta la zona anterior a las orejas sin incluirlas. Tanto el cráneo como las orejas estarán ya incluidos en el casco realizado en el otro proyecto paralelo. Por consiguiente, la primera acción ha sido eliminar la parte superior y trasera de la cabeza del avatar. En modo edición, se seleccionan los vértices correspondientes a la parte trasera y superior del cráneo y se eliminan de manera que se obtiene únicamente el rostro.

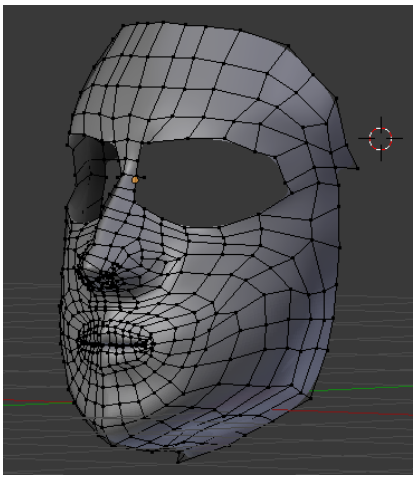


Figura 6.1- Malla del rostro del avatar reutilizada, para modelar la máscara.

En la construcción del avatar, la cabeza estaba dividida en varios objetos separados para facilitar su edición: por un lado los ojos, interior de la boca y por último la propia cabeza. La ventaja de haber separados los objetos previamente, es que a continuación se pueden eliminar aquellos que no se vayan a utilizar, a diferencia de si se tuviese todo en un mismo objeto (en ese otro caso habría que ir seleccionando y eliminando manualmente los vértices correspondientes). Una vez eliminados los dos objetos que son cada uno de los ojos, resulta la malla con las dos cavidades pertenecientes a estos, como en un antifaz. Para tapar y mallar las dos cavidades se emplea el comando de unión de vértices. Al seleccionar tres o más vértices de los bordes de las cavidades se generará una cara nueva con sus aristas. Repitiendo este proceso se consigue cubrir la totalidad de las cavidades de la zona ocular. La misma operación se realizará para las cavidades de la zona nasal, hasta conseguir que todo el rostro quede unido en una misma malla. El resultado final se muestra en la figura 6.2.

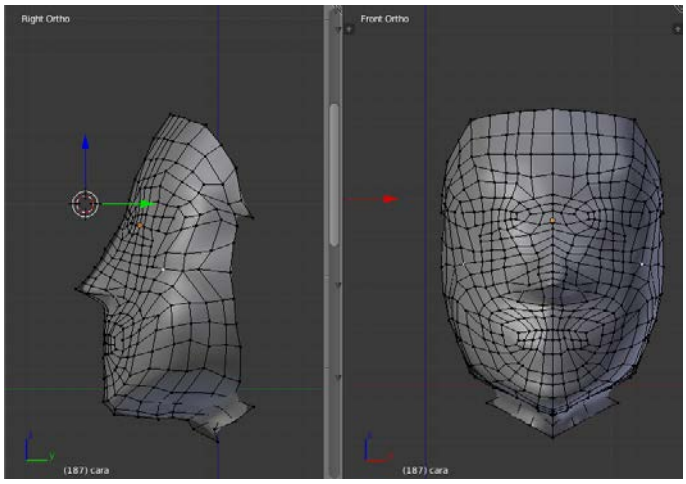


Figura 6.2- Máscara modelada

Es importante que no quede ninguna cavidad o hueco en la malla de la máscara, ya que se va a realizar la impresión 3D de la máscara como un único objeto. Se ha decidido no incluir elementos faciales como la forma de los labios, ya que todos los rasgos faciales ya los lleva implementado el propio avatar. Si se incluyese la forma labial en la máscara, y por ejemplo el avatar abriese la boca, se generaría un efecto extraño, como ocurría con el primer prototipo del *Mask-bot* (apartado 2.3.2). Tras haber obtenido la forma de la máscara hay que evitar las protuberancias de la malla y conseguir que el aspecto superficial de la máscara quede lo más suave posible. Es normal que en el objeto salgan pequeños abultamientos en el diseño ya que la malla en sí está compuesta por formas poligonales a partir de la unión de los vértices. Para suavizar la superficie de la malla, existen varios procedimientos.

En concreto, se puede ir modificando manualmente la posición de los vértices en la superficie. Esta tarea resulta más tediosa, y es poco recomendable salvo si existen muy pocas protuberancias. Al modificar la posición de un vértice, las aristas que lo unen a los vértices vecinos cambian de posición, lo cual puede generar nuevas protuberancias en los alrededores. De forma general, es más efectivo el uso de la herramienta *smooth vertex.* Esta herramienta se encarga de suavizar los vértices del objeto promediando el ángulo entre las caras vecinas. Para realizar esta operación, en modo edición se selecciona el conjunto de vértices que se desean suavizar y a continuación se emplea esta herramienta apropiada (situada en la barra de herramientas, herramientas de malla, en el sub-apartado deformar). Si se realizan muchas iteraciones de esta operación, la superficie de la malla puede quedar tan lisa que se pierda el aspecto original de malla. En este caso se ha optado por realizar dos iteraciones, ver figura 6.3.

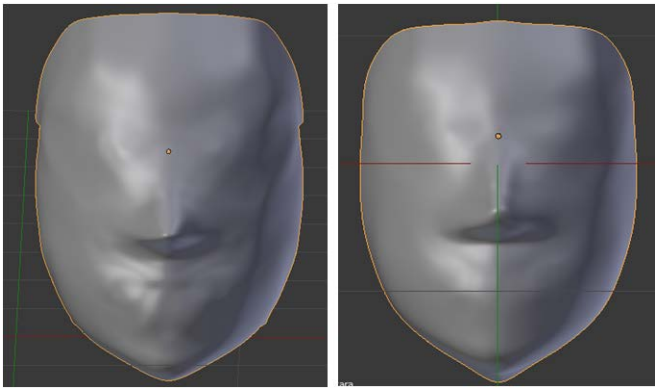


Figura 6.3 - Eliminación de la rugosidad superficial

Una vez terminado el modelado, el último paso consiste en escalar la máscara a una dimensión óptima para que más adelante se acople al casco sin inconvenientes. El objetivo es obtener una máscara de dimensiones similares a un rostro humano, por lo que el objeto ha sido escalado hasta obtener las siguientes dimensiones:

-Anchura del rostro: Distancia recta vertical desde la barbilla hasta parte superior de la frente →20 cm.

-Longitud horizontal del rostro: Distancia recta horizontal entre extremos de la máscara, desde donde comenzarían las orejas. →15cm.

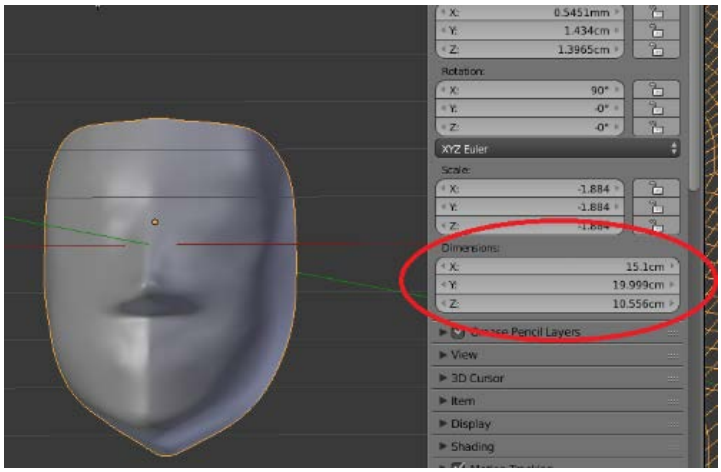


Figura 6.4 - Escalado de la máscara

En un principio se planteó que la unión mecánica entre máscara y casco consistiese en atornillar la máscara a los bordes del casco. De esta manera la máscara quedaría bien fijada al casco. El problema que se ha observado, es que si el contorno del casco y el contorno de la máscara difieren en las dimensiones, o si la máscara no se coloca centrada

en el casco, habría que cambiar la dimensión de la máscara, imprimirla otra vez o atornillar de nuevo en otra posición. Por este motivo, se ha propuesto otra forma de unir la máscara y el casco. Esta unión consiste en sobredimensionar ligeramente los bordes de la máscara de modo que esta se pueda encajar dentro del casco por medio de una ranura. Al ser el material de impresión de la máscara, un material más flexible que el del casco, la máscara entra en la ranura creada en el casco y queda bien fijada.

En el trabajo Fin de Grado que se está desarrollando de forma concurrente con el actual se ha diseñado la forma de la ranura, o lo que es lo mismo, el hueco del casco donde se encajará la máscara. Este espacio es de 2.8 mm, que coincide con el grosor de la máscara. Esta ranura ha sido implementada utilizando el programa Catia.

Figura 6.5- Forma de la ranura que se implementará en el casco para que la máscara se incruste en ella

El archivo .stl de la ranura se ha transformado al formato .blend. Finalmente, desde Blender, se han modificado los bordes de la máscara y se ha sobredimensionado para que se ajuste a la forma de la ranura y encaje perfectamente en el casco. En la figura 6.6, el aro amarillo corresponde con la ranura entre la máscara y el casco. Con este método, la máscara queda bien encajada a presión en el casco, y también de esta forma es más sencillo el quitar y poner la máscara.

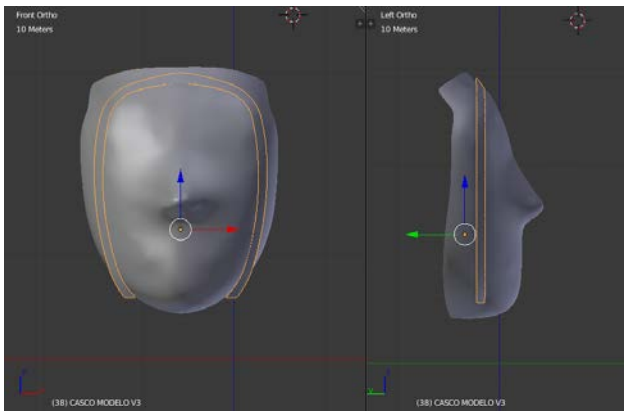


Figura 6.6- Ajuste de la máscara a la forma de la ranura

6.3. Fabricación de la máscara

Una vez actualizado el modelo Blender de la máscara conforme al contorno de la ranura, el archivo se ha transformado a formato .stl. El formato .stl es el que define la geometría de los objetos 3D, en este caso la máscara, y será el formato leído por la aplicación de la impresora. Para el proceso de impresión se ha empleado una impresora 3D marca *Ultimaker S3*. Previo a la impresión, el archivo .stl se envía a la aplicación *Ultimake Cura*. Este programa de la impresora, se encarga de dividir el archivo del modelo en distintas capas y es el que genera el lenguaje de impresión código .gcode para que la impresora pueda interpretarlo. Después de este proceso, el archivo .gcode con los movimientos de impresión y coordenadas, se envía a la impresora 3D por medio de un USB para que la impresora lo procese. Los principales parámetros de la impresión son:

- Material de impresión de la máscara: PLA transparente.
- Espesor de capa: 0.1mm. Es el espesor más fino que se puede conseguir con un extrusor de diámetro 0.4mm.
- Material del soporte para mantener la máscara en la impresión: PVA. PVA es un material soluble en agua para evitar marcas en la pieza al retirar los soportes una vez finalizada la impresión.
- Porcentaje de relleno de la impresión: 100%, para evitar zonas con patrones generado por los filamentos.
- Espesor final de la máscara: 2.8mm.

En la figura 6.7 se muestra el resultado de la máscara.

Figura 6.7- Máscara impresa

6.4. Diseño del sistema de retroproyección

Antes de establecer un método para la retroproyección de la imagen, se han de tener presentes ciertas características del proyector, como el tamaño de imagen o el ratio de aspecto de este. Por ello el primer paso es seleccionar el proyector que se adecue a las condiciones del modelo. Tras esto ya se puede realizar un análisis para encontrar que método de proyección es más adecuado.

6.4.1. Elección del proyector

Previo a entrar en la búsqueda de distintos proyectores, se van a evaluar las características principales que debe poseer el proyector elegido acorde a las necesidades del presente trabajo:

- El proyector tiene que ser lo más compacto posible, ya que por un lado no ha de ser visible cuando el sistema este montado y por otro lado es importante que se ubique en una posición que no interfiera con la parte mecánica de la cabeza, especialmente a la hora de montar y desmontar las partes del casco. Teniendo en cuenta estas condiciones, un mini-proyector resulta la opción más recomendable.
- El objetivo o lente del propio proyector debe estar, a ser posible, centrado en el punto medio en la cara en la que esté ubicado. De esta forma es más fácil tener una referencia de colocación para que la imagen proyectada quede centrada.
- El ratio de aspecto del proyector es recomendable que sea 16:9 o 4:3. El ratio de aspecto mide la proporción entre la anchura y altura de la imagen. El avatar que aparecerá en la imagen del proyector debe ajustarse a las dimensiones de la máscara, y la relación de altura y anchura de la máscara no es 1:1. Al igual que el tamaño de un rostro real, la altura de la máscara es un poco mayor que su anchura.
- El proyector debe poseer una entrada HDMI, para conectarlo a un dispositivo o pantalla externa desde donde se controlará al avatar.

Después de la búsqueda y análisis de varios modelos se ha procedido a seleccionar el siguiente proyector: *Mini proyector marca ExquizOn modelo S6* (figura 6.8). Las principales características de este proyector se muestran en la tabla 10.

Dimensiones del producto	5.5 x 5.5 x 5,5 cm
Relación de aspecto	16:9
Resolución de pantalla	854 x 480
Máxima resolución de pantalla	1080p Full HD
Tipo de pantalla	DLP
Tasa de contraste	1000:1
Brillo de imagen	100:1
Otros componentes incluidos	Mando a distancia o control remoto, conector micro HDMI, 1 cable de alimentación para cargar la batería y manual de instrucciones

Tabla 10 – Características del proyector

Figura 6.8- Imagen del mini proyector marca ExquizOn modelo S6
Imagen extraída de la web: https://es.gearbest.com/projectors/pp_009202819836.html

La razón por la que se ha elegido este mini proyector es que cumple con los requisitos previamente citados. En cuanto al tamaño, es un proyector muy compacto en las tres dimensiones. El problema que se encontraba en otros proyectores es que el largo superaba los 15-20 cm en la mayoría de los casos. Respecto a la ubicación del objetivo del proyector, aunque este no se encuentre exactamente centrado en la cara, tampoco se encuentra próximo a ninguno de los bordes, lo que permite una mejor resolución del problema. También cumple con el ratio de aspecto deseado, y consta de una entrada micro HDMI.

De forma adicional, el producto cuenta con tecnología DLP. Los proyectores con la tecnología DLP constan de un chip denominado DMD que está formado por una matriz de microespejos. Cada microespejo representa cada pixel de la imagen, y el total de la matriz de espejos es proporcional a la resolución o la relación de aspecto de la imagen

proyectada. Para la formación de las imágenes, cada uno de los microespejos rota entre dos ángulos distintos: Uno de los ángulos proyecta un pixel oscuro, y el otro ángulo proyecta un pixel brillante. La escala intermedia de gris se produce rotando todos los espejos a gran velocidad entre ambos ángulos. Para generar el color de la imagen en muchos proyectores de este tipo, se coloca una rueda de color RGB entre una fuente de luz y la matriz de microespejos DMD. Esta rueda gira a alta velocidad frente a la fuente de luz que atraviesa la rueda, y es proyectada hacia la matriz. De esta forma se generan los ciclos de color sobre los microespejos de la matriz. Finalmente, los microespejos reflejan la luz y con una lente interna se proyecta el color sobre cada uno de los pixeles de la imagen, donde cada pixel de la imagen corresponde a cada microespejo de la matriz. En algunos casos de proyectores DLP como este, en lugar de utilizar una lámpara de luz, se emplean LED (diodos emisores de luz), donde no se precisa de rueda de color para efectuar el ciclo de los colores, realizando este cometido electrónicamente.

Otro de los aspectos de esta tecnología, es que con la utilización del chip de microespejos, se evita la degradación del color después de muchas horas de uso como si ocurre con proyectores de tecnología LCD. Esta degradación se produce por el calor irradiado de la fuente de luz que quema el LCD (pantalla de cristal líquido). Sin embargo los microespejos del chip DMD no son afectados por el calor y temperatura de la fuente.

Respecto al brillo de la imagen y la tasa de contraste del proyector, no son especificaciones que requieran excesiva exigencia para este trabajo. Por un lado la luminosidad del proyector es de 100 lúmenes. Para cuantificar la luminosidad hay que tener en cuenta el tamaño de imagen que se desea obtener. Esto es así porque la potencia luminosa se reparte entre la superficie de la pantalla y cuanto mayor sea la pantalla o imagen proyectada, más potencia luminosa se requiere. Sin embargo la imagen que se va a proyectar no va a ser de gran tamaño, sino de un tamaño acorde a la máscara para proyectarla sobre ésta. Es por eso que no se requiere de una potencia luminosa alta para visualizar la imagen correctamente, incluso en habitaciones con luz.

Por otro lado, la tasa de contraste es de 1000:1. Esto significa que el pixel más brillante es 1000 veces más luminoso que el pixel más tenue. Al igual que con el brillo, el contraste no es una característica exigente para este proyecto. La imagen proyectada va a ser el rostro del avatar, y el nivel de contraste no va a afectar a la visualización de sus distintas partes.

Los principales inconvenientes que se han encontrado en el proyector, son el factor de proyección (*TR*), la duración de la batería y la carencia de control remoto para el encendido y apagado. El ratio de disparo o factor de proyección, es un factor que mide la relación entre la distancia de proyección y el ancho de la imagen proyectada. Este concepto se tratará en el apartado 6.4.2. En relación a la duración de la batería, esta duración oscila entre 1-2 horas. Por último, el proyector consta de un mando a distancia para manejar las opciones, si bien el encendido y apagado es manual.

En cuanto al precio del producto, es un precio estándar de gama media. Hay proyectores en el mercado de más alta calidad y precio, pero con el presente modelo se consigue un compromiso entre coste y prestaciones adecuado a los requisitos y condiciones iniciales para las cuales se inició la búsqueda.

6.4.2. Sistema de proyección empleado

En este apartado se va a exponer el procedimiento seguido para determinar el sistema de proyección más adecuado conforme a los recursos disponibles. Hay que mencionar que se han intentado implementar varios métodos de proyección, hasta que finalmente se ha obtenido un sistema óptimo donde la imagen queda bien proyectada sobre la máscara. Se van a evaluar todos estos métodos y el porqué de su implementación. La simbología utilizada a lo largo de este apartado y en los cálculos es la siguiente:

- *w*: Anchura de la máscara.
- *h*: Altura de la máscara.
- *t*: Tamaño de la imagen de la máscara. $t = \sqrt{w^2 + h^2}$
- *W*: Anchura de la imagen proyectada.
- *H*: Altura de la imagen proyectada.
- *T*: Tamaño de la imagen proyectada. $T = \sqrt{W^2 + H^2}$
- *D*: Distancia de proyección, desde el objetivo del proyector a la imagen proyectada.
- *D'*: Distancia de proyección, desde la lente (acoplado al proyector) a la imagen proyectada.

En primer lugar hay que analizar la finalidad y los datos disponibles. La finalidad es proyectar la imagen sobre la máscara. En cuanto al tamaño de la máscara, se ha optado porque tenga un tamaño similar al de un rostro humano adulto. Debido a que la máscara no va a ser plana, sino que va a presentar cierto relieve en la zona de la nariz principalmente, se pueden obtener dos dimensiones de la máscara según la forma de medir (figura 6.9):

-Medida recta de la máscara: 15cm x 20 cm ($w \times h$).

-Medida en relieve de la máscara: 20cm x 23 cm ($w \times h$).

Considerando que la anchura se mide en el eje X y la altura se mide en el eje Y, la medida en relieve de la máscara se ha calculado según se explica a continuación. Para medir la altura se ha cogido el punto del eje X que divide la máscara en dos mitades simétricas. Esta medida abarca la distancia desde la zona del mentón o barbilla hasta la parte superior de la frente, incluyendo la curvatura de la nariz. Para medir la anchura se ha cogido el punto del eje Y que divide la máscara en dos mitades iguales no simétricas, que es el tren superior y el tren inferior del rostro. Esta medida abarca la distancia entre los extremos de la máscara donde comenzarían las orejas, incluyendo la curvatura de la nariz. La medida recta de altura y anchura se mide igual sin considerar la curvatura de la máscara.

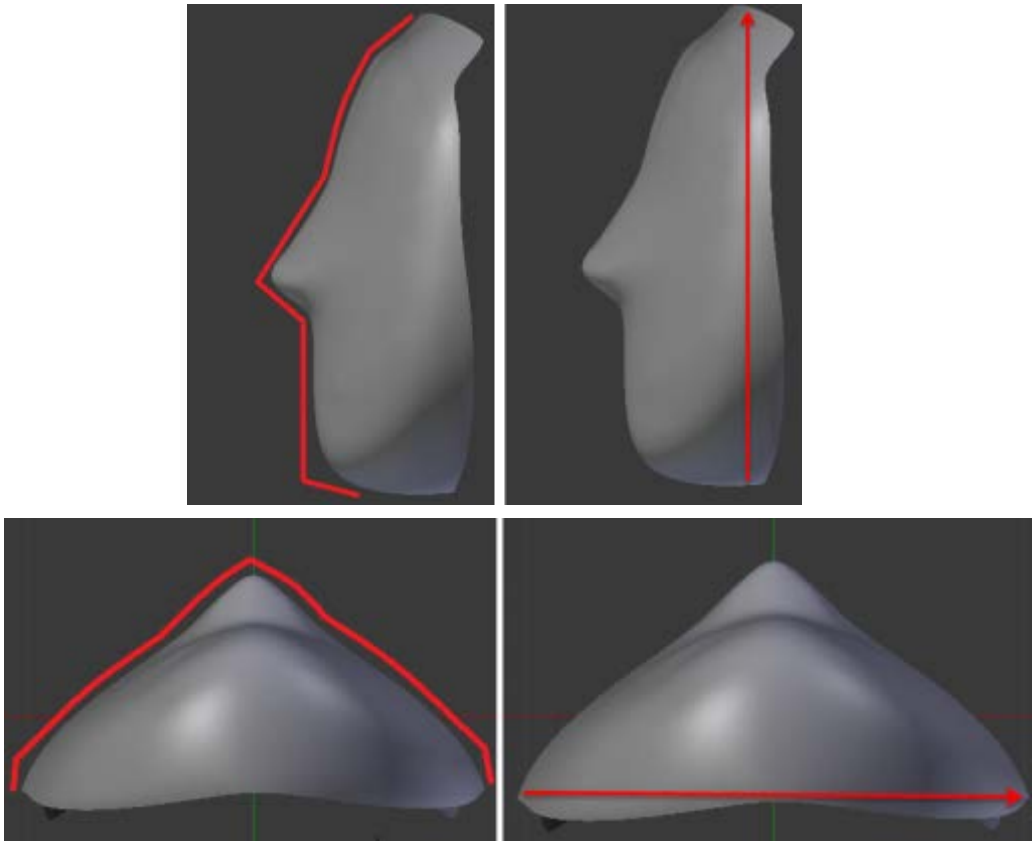


Figura 6.9 – Comparativa de la medida en relieve y medida recta de la altura y anchura de la máscara

La dimensión que se va a tener en cuenta a partir de ahora es la obtenida en la medida en relieve (20cm×23cm) que es la medida real ya que incluye el factor de la curvatura. Si se tuviese en cuenta la medida recta, la imagen proyectada no iba a cubrir todo el tamaño de la máscara. En la tabla 11 se exponen los datos disponibles del manual del proyector referidos a la relación entre el tamaño de imagen *T* y su correspondiente distancia *D*.

Tamaño de imagen (*T*)	Distancia de proyección (*D*)
7 pulgadas =17.78 cm	0.18m
38 pulgadas=96.52 cm	1m
56 pulgadas=142.24 cm	1.5m
100 pulgadas=254 cm	2.5m

Tabla 11 – Relación del tamaño de la imagen con la distancia de proyección

Ratio de aspecto, *Ra*: La proporción entre la anchura y altura de una imagen. $Ra = W/H$. El ratio de aspecto del proyector es de **16:9**. Esto significa que la anchura de la imagen tiene una longitud 1.78 veces mayor que la altura de la imagen. El formato de la máscara es al contrario, la anchura es menor que la altura y con distinta proporción que la dada por el ratio de aspecto del proyector. Es por eso que hay que girar el proyector para que la imagen del proyector quede vertical, y la altura sea mayor que la anchura.

Factor de proyección, *TR*: Es la relación entre la distancia de proyección D, y el ancho de la imagen W. $TR = D/W$.

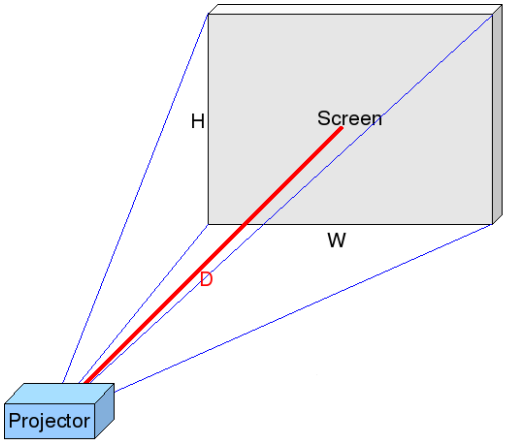


Figura 6.10 – Ejemplo de proyección de imagen de anchura W a una distancia D
Imagen extraída de la página web: https://es.wikipedia.org/wiki/Factor_de_proyecci%C3%B3n

Conociendo el tamaño de la imagen *T* y el ratio de aspecto *Ra*, se puede obtener tanto la altura *H* como la anchura *W* de la imagen. Para cada uno de los datos *T* de la tabla 11 del manual, se han calculado los correspondientes *W*. Como además se conoce la distancia *D* para cada medida, después se ha calculado el *TR* del proyector. A continuación se muestra el cálculo de los valores *W* de la tabla, y sus *TR*.

-Para *T* = 17.78 cm → $17.78 = \sqrt{W^2 + H^2} = \sqrt{(1.78H)^2 + H^2} \rightarrow H = 8.716cm.$

$$W = 1.778H = 15.49cm\ , \qquad TR = \frac{D}{W} = \frac{18}{15.49} = 1.16 \tag{8}$$

-Para *T* = 95.52 cm → $95.52 = \sqrt{W^2 + H^2} = \sqrt{(1.78H)^2 + H^2} \rightarrow H = 47.32cm.$

$$W = 1.778H = 84.13cm\ , \qquad TR = \frac{D}{W} = \frac{100}{84.13} = 1.18 \tag{9}$$

-Para *T*=142.24 cm → $142.24 = \sqrt{W^2 + H^2} = \sqrt{(1.78H)^2 + H^2} \rightarrow H = 69.73cm.$

$$W = 1.778H = 123.98cm\ , \qquad TR = \frac{D}{W} = \frac{150}{123.98} = 1.20 \tag{10}$$

-Para *T* = 254 cm → $254 = \sqrt{W^2 + H^2} = \sqrt{(1.78H)^2 + H^2} \rightarrow H = 124.52cm.$

$$W = 1.778H = 221.39cm\ , TR = \frac{D}{W} = \frac{250}{221.39} = 1.13 \tag{11}$$

Con los datos del manual, se ha obtenido un valor de *TR* de 1.1-1.2 aproximadamente. Es decir, este proyector tiene un factor de proyección estándar. Para casos donde se desea obtener una imagen grande a largas distancias como en un aula, cuanto mayor sea el *TR* mejor. Sin embargo lo ideal para el caso de este proyecto, sería que el proyector tuviese un disparo corto o ultra corto, *TR*<1. Porque se desea obtener una imagen considerable a una distancia muy cercana. Es por eso que el *TR* se citó como uno de los inconvenientes del proyector en el apartado anterior.

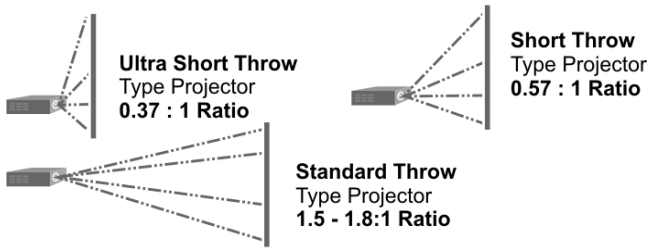


Figura 6.11 –Tipos de proyectores según el factor de proyección
Imagen extraída de la página web: http://projectionscreensolutions.co.uk/selecting-a-screen/

En los siguientes apartados se va a calcular la distancia de proyección *D* necesaria, y se hará un análisis para buscar el sistema de proyección más óptimo.

6.4.2.1. Proyección de la imagen vertical sobre la máscara

Primero se va a analizar la posibilidad de realizar la proyección de la imagen vertical sobre la máscara sin ningún otro elemento, para ver la *D* necesaria y si esta distancia es considerable o no. Inicialmente hay que calcular el tamaño de la imagen proyectada para el tamaño de la máscara. Como se ha girado el proyector para que la imagen quede en vertical, ahora la $H = 1.78 \times W$, debido al ratio de aspecto del proyector mencionado anteriormente.

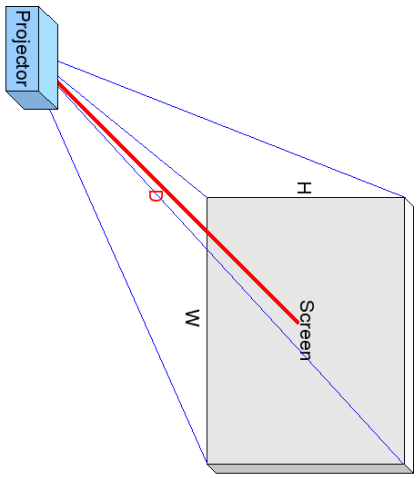


Figura 6.12 – El giro del proyector implica que la altura ahora es la dimensión mayor
Imagen extraída de la página web: https://es.wikipedia.org/wiki/Factor_de_proyecci%C3%B3n

Entre la anchura y la altura de la máscara, la anchura *w* es la dimensión menor. Por lo tanto se tomará la w como valor límite, y se va a considerar que *W=w* para los cálculos. Es decir, el ancho de la imagen proyectada ha de ser igual al ancho de la máscara. Teniendo en cuenta esta condición, y el ratio de aspecto del proyector, la altura de la imagen proyectada H para este valor de W es:

$$W = w = 20\ cm \tag{12}$$

$$H = \frac{16}{9} \times 20 = 35.56\ cm \tag{13}$$

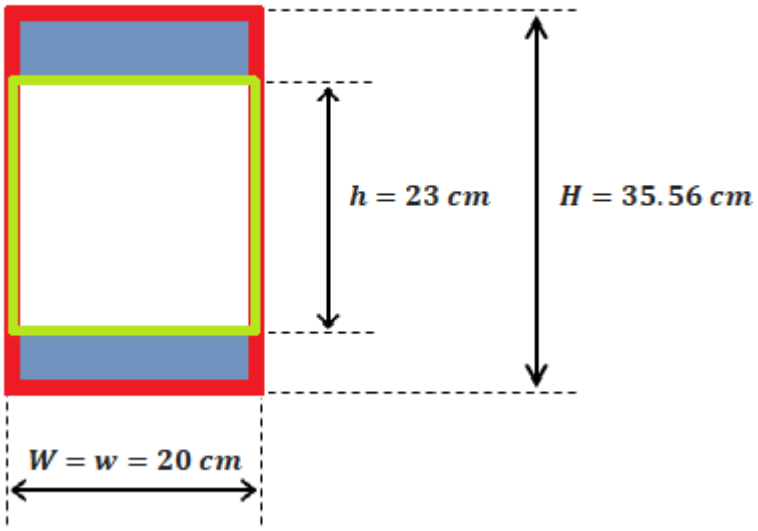


Figura 6.13 - Dimensiones de la imagen proyectada (rojo) teniendo en cuenta la relación 16:9 del proyector, sobre las dimensiones de la máscara (verde).

Para que la imagen proyectada se ajuste a la máscara, la dimensión límite es la anchura, que tiene que ser 20 cm. Con esta medida, la altura de la imagen no supone una limitación, porque la altura de la máscara es inferior a la altura máxima de la imagen proyectada, ver figura 6.13. Como la imagen tiene que estar centrada en el punto medio de la máscara, va a existir un área por encima y por debajo de la máscara. Esta superficie sobrante (zona azul de la figura 6.13), se proyectará sobre el casco que es opaco, por lo que no será visible desde el exterior. El resto de la imagen se proyectará sobre el área correspondiente a la máscara translucida por lo que se visualizará externamente.

A continuación, se va a proceder al cálculo de la distancia de proyección *D* necesaria para obtener el tamaño de imagen *T* definido por su anchura y altura. En concreto, este tamaño es el siguiente:

Para *W*=20cm y *H*=35.56cm → $T = \sqrt{W^2 + H^2} = \sqrt{20^2 + 35.56^2} \rightarrow T = 40.80cm$ (14)

Ahora hay que obtener la distancia *D* correspondiente a este tamaño de imagen *T*. Como son conocidos otros valores de *T* y su distancia *D* por los datos del manual del proyector (tabla 11) la opción más óptima es realizar una interpolación lineal. Antes se ha de verificar la linealidad de la función dada por los valores de *T* y *D*.

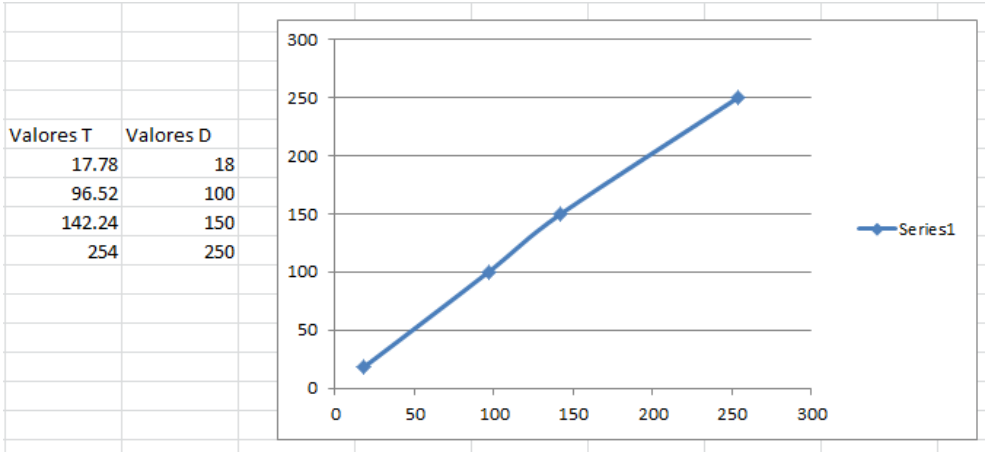


Figura 6.14 – Relación entre T y D de los datos obtenidos del manual del proyector

En la figura 6.14 se observa que la función es aproximadamente lineal en todo el rango de datos, con una mayor desviación en la zona comprendida entre los dos últimos valores. No obstante, dado que *T* está comprendido entre el valor 17.78cm y 96.52cm (ambos valores situados en la zona lineal), se puede emplear el método de la interpolación lineal para calcular el valor de *D* sin perder precisión en el resultado. En la siguiente ecuación se muestra el cálculo de *D* para el tamaño de imagen *T*=40.80 cm.

$$\frac{96.52 - 17.78}{100 - 18} = \frac{96.52 - 40.80}{100 - D} \rightarrow D = 41.96\ cm \qquad (15)$$

Para realizar la proyección de la imagen en vertical centrada en el punto medio de la máscara, se necesitaría una distancia *D*=41.96 cm a fin de que la imagen proyectada encajase en la máscara con las medidas correspondientes, tal y como se muestra en la figura 6.13. Sin embargo, esta distancia de proyección es demasiado elevada para poder ubicar el proyector dentro de la cabeza. Se necesitaría realizar un casco excesivamente grande para ubicar el proyector en su interior. Por consiguiente, se van a buscar otras soluciones para reducir la distancia de proyección.

6.4.2.2. Proyección con ayuda de un espejo

Un sistema de proyección que permite reducir la distancia *D*, consiste en utilizar un espejo que refleje la imagen del proyector sobre la máscara. Este método fue empleado en un proyecto de diseño de un sistema robot orientado a aplicaciones de telepresencia [22]. La configuración del sistema se basa en colocar el proyector con el objetivo orientado hacia arriba, y colocar un espejo con un ángulo de 45º a una distancia D_e del proyector. D_e es la distancia entre el proyector y el espejo. Manteniendo este ángulo del espejo, se han realizado un conjunto de pruebas para distintos valores de D_e . Para cada valor de D_e se ha ido modificando también la distancia *D*. El problema principal que se ha encontrado con este método es que para el tamaño del espejo dado, la distancia *D* tiene que ser excesivamente grande a fin de que la imagen abarque el tamaño de la máscara. Para una *D* corta, la imagen queda muy pequeña. Con la distancia D_e ocurre lo mismo. También se ha probado a variar el ángulo del espejo, pero este parámetro no cambia el tamaño de la imagen proyectada, solo orienta la imagen más arriba o más abajo en la máscara, ver figura 6.15. Existen también otros factores que afectan a la proyección de la imagen, como es el tipo de espejo y su tamaño. En este caso la geometría del espejo disponible era circular con un diámetro de 13cm.

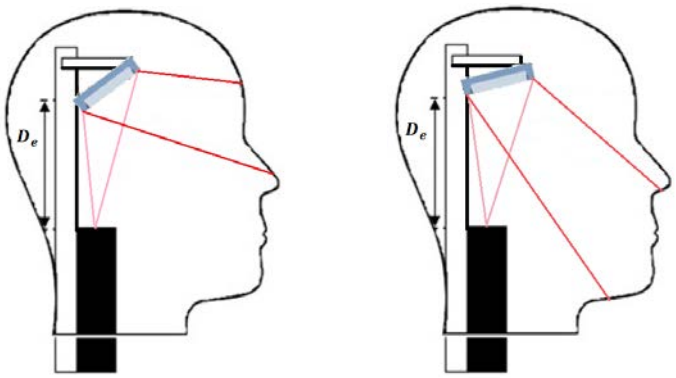


Figura 6.15 – Proyección de imagen en el espejo reflejada sobre el rostro, para dos ángulos distintos y una D_e y D asumibles.

Las dos posibles soluciones para corregir esta cuestión, sin modificar la distancia *D*, es o aumentar el tamaño del espejo o emplear un espejo de tipo divergente.

- La solución de aumentar el tamaño del espejo resulta imposible ya que hay que colocar el casco encima, y colocar un espejo de mayor dimensión al utilizado hasta ahora no entraría bajo el casco.

- El espejo que se ha empleado no es ni convergente ni divergente, esto hace que la imagen reflejada sea del mismo tamaño que la imagen que llega al espejo. Con la utilización de un espejo divergente se puede arreglar el problema. El concepto de divergencia en los espejos explica el modo en cómo se reflejan los rayos entrantes. Los rayos de luz que llegan a la superficie del espejo de forma recta, se divergen durante el reflejo. El punto focal estaría detrás del espejo, y estos rayos reflejados tienden a separarse o divergir. Aplicado al presente caso, la imagen reflejada sobre la máscara aumentaría de tamaño, que es lo deseado para cubrir el tamaño de la máscara.

Ambas soluciones finalmente han resultado inviables para los recursos disponibles. Por consiguiente no se ha empleado este método y se ha buscado otro sistema de proyección más factible.

6.4.2.3. Proyección de la imagen vertical añadiendo una lente al proyector

El último sistema de proyección que se ha considerado utilizar, es la proyección de la imagen en vertical utilizando el proyector junto a una lente externa. La lente irá acoplada al objetivo del proyector, mediante un imán. La lente utilizada es una lente ojo de pez de 180º grados.

Figura 6.16 - Imagen del proyector y lente utilizados

Para una misma distancia, el uso de una lente externa ayuda a que la imagen proyectada tenga más tamaño respecto a si solo se utilizase el proyector. Con el uso de la lente más el proyector, el factor de proyección disminuye ya que la *W* aumenta para una mismo *D*. De modo que el proyector que en un principio tiene un *TR* estándar, se "convierte" en un proyector de *TR* corto. A pesar de esto, el uso de lentes puede provocar algunos inconvenientes en la imagen que son los siguientes:

-Aberración cromática: Es un efecto que provoca la aparición de bordes coloreados en los objetos de la imagen proyectada debido a la incapacidad de la lente de enfocar todos los colores hacia un mismo punto o foco. En el presente trabajo este fenómeno no se ha dado, posiblemente porque la proyección se realiza a una distancia relativamente cercana, y la desviación del enfoque de los colores no resulta significativa.

-Distorsión: Este fenómeno se produce cuando la imagen, a pesar de aumentar de tamaño gracias a la lente, se distorsiona respecto a la imagen original. En el presente trabajo, se ha comprobado que la imagen proyectada sobre una superficie plana sí que se distorsiona en la zona de las esquinas de la imagen. La distorsión se puede corregir deformando la superficie proyectada, ver figura 6.17. Como la superficie final de proyección en este caso va a ser la máscara y esta es una superficie no plana, el problema de la distorsión termina corrigiéndose. La distorsión en las esquinas queda compensada por la curvatura en la zona del contorno de la máscara.

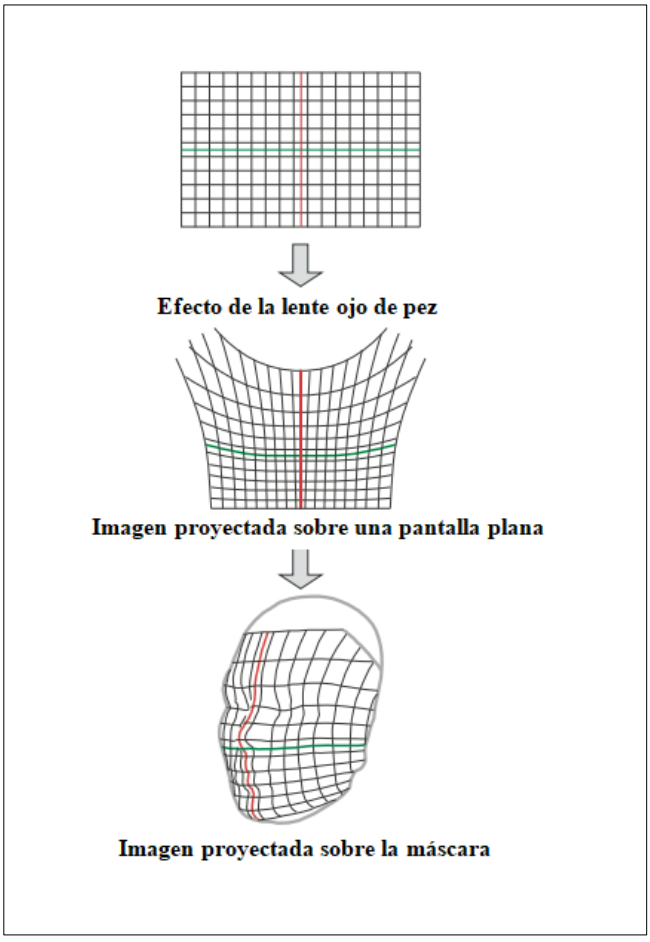


Figura 6.17 –Corrección de la distorsión de la imagen, mediante una superficie deformada.

Ahora hay que calcular la distancia *D´* exacta para que la imagen encaje exactamente en la máscara. *D´* es la distancia de proyección utilizando el conjunto proyector más lente. Lo único conocido es que la *D´* < *D*, ya que se va a necesitar menos distancia empleando la lente para conseguir una imagen de mismo tamaño que la imagen obtenida a una distancia *D*.

La forma para calcular la nueva distancia *D´* va a ser mediante trigonometría. Si se conoce el ángulo de proyección y la *W*, es posible obtener la distancia de proyección deseada.

Calculo del ángulo de proyección (solo proyector):

Para cualquier distancia *D* aunque la *W* de la imagen cambie, el ángulo de proyección va a ser siempre el mismo. Cogiendo una distancia cualquiera por ejemplo la distancia *D*=41.96cm calculada en el apartado 6.4.2.1. , se calcula el ángulo de proyección.

$$\tan\theta = \frac{\frac{W}{2}}{D} = \frac{10}{41.96} \rightarrow \boldsymbol{\theta = 14.90\,^{\circ}} \qquad (16)$$

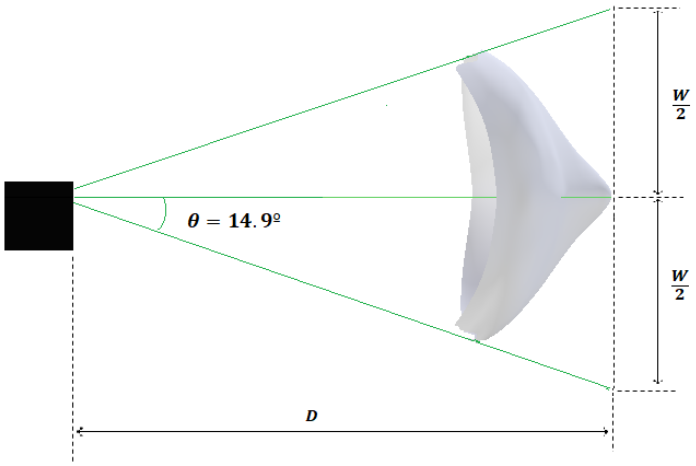


Figura 6.18 – Esquema de la proyección sobre la máscara, visto desde la planta

Cálculo del ángulo de proyección (proyector más lente):

Para esa misma distancia *D*=41.96cm, se va a acoplar la lente para determinar experimentalmente cual es el nuevo tamaño de *W'* y posteriormente calcular el nuevo ángulo de proyección. La *W'* obtenida experimentalmente es *W'*=41.13cm. Para este nuevo valor de *W'* se calcula el ángulo θ':

$$\tan\theta = \frac{\frac{W}{2}}{D} = \frac{\frac{41.13}{2}}{41.96} \rightarrow \boldsymbol{\theta' = 29.01\,°} \tag{17}$$

La *W* límite para que la imagen entre en la máscara era de 20cm.

$$D' = \frac{\frac{W}{2}}{\tan\theta'} = \frac{\frac{20}{2}}{\tan(29.01°)} = 20.40cm \tag{18}$$

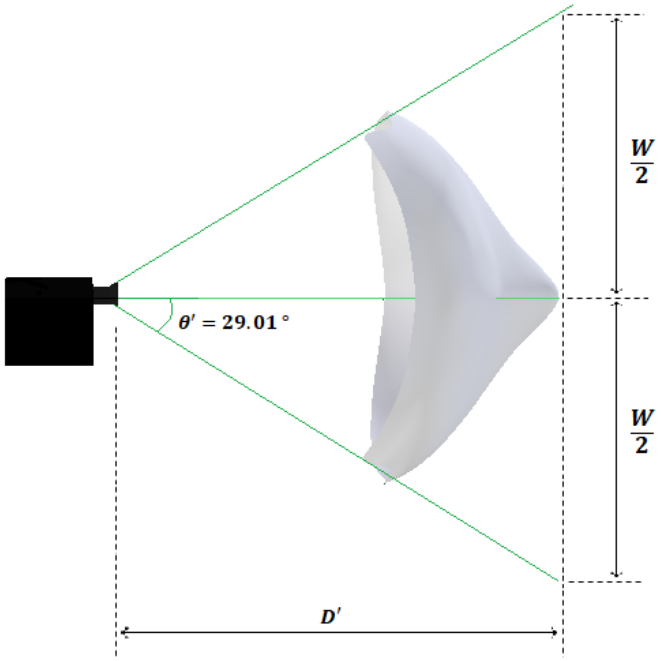


Figura 6.19 – Esquema de la proyección con la lente sobre la máscara, visto desde la planta

Para corroborar los cálculos, se ha probado a colocar el proyector con la lente a la distancia *D´*=20.4 cm y la distancia *W* resultante es la correcta, *W*=20cm. Es importante que el objetivo del proyector este centrado respecto a la máscara para que la imagen quede también centrada en la máscara. La distancia de proyección se ha reducido prácticamente la mitad, de un valor inicial *D*=41.96cm a un valor *D´*=20.4cm con ayuda de la lente acoplada al proyector.

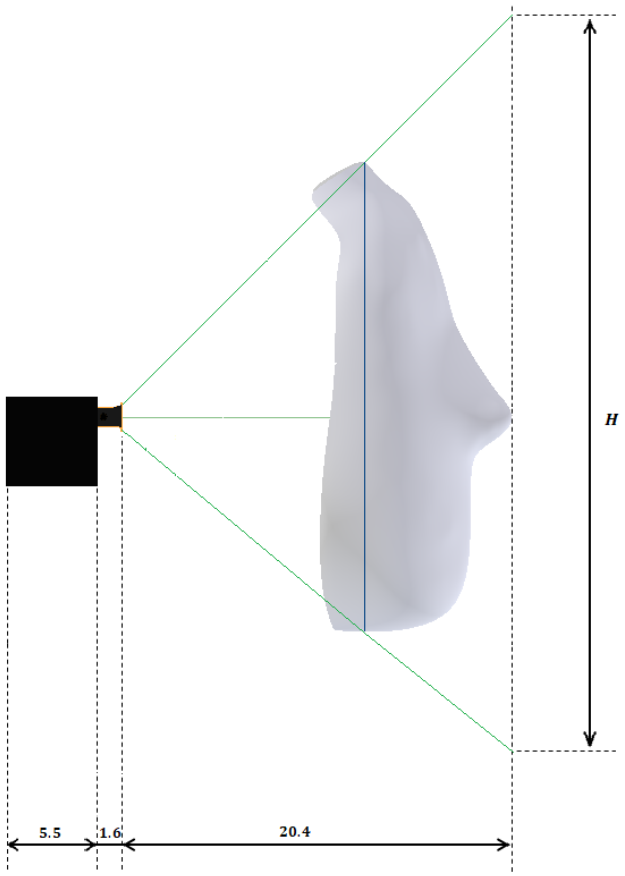


Figura 6.20 – Esquema de distancia de proyección desde el proyector con la lente hasta el punto final de la máscara

La distancia total desde la imagen proyectada en la máscara, hasta la parte trasera del proyector es:

$$dtotal = D' + L_L + L_P = \ 20.4\ cm + 1.6cm + 5.5cm = 27.5\ cm \qquad (19)$$

L_L = *Largo o profundidad de la lente.*
L_P = *Largo o profundidad del proyector.*

Después de elegir el sistema de proyección y distancias, en el proyecto concurrente se ha procedido a construir el casco y la zona donde irá encajado el proyector. El proyector estará alojado en un cubículo, en la parte trasera de la cabeza con una abertura por detrás para poder encender/apagar el aparato y conectar el cable HDMI. En la parte delantera se acopla la máscara en una ranura realizada en el borde del casco.

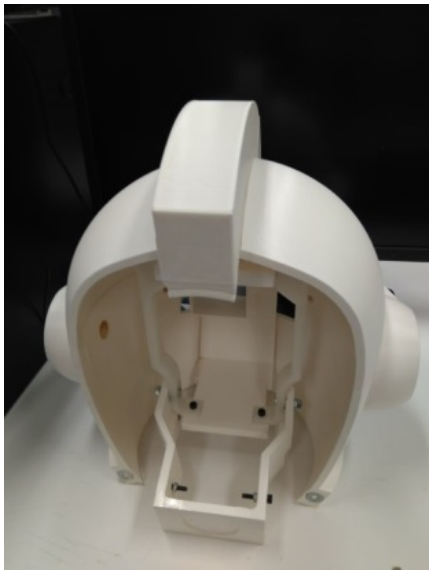


Figura 6.21 – Interior del casco donde se ubicará el proyector

Capítulo 7 - Experimentación y análisis de resultados

7.1. Introducción

En el presente capítulo se van a mostrar los resultados de la proyección del avatar sobre la máscara unida ya al casco del robot, y se observará la apariencia final que tiene el rostro. También se presentarán los resultados de la ejecución de las expresiones faciales en el avatar después de realizar una simulación utilizando el gestor de comportamiento que se explicó en el capítulo 4. Por último se mostrará la ejecución del sistema de sincronización y voz y se sacarán conclusiones al respecto.

7.2. Configuración del sistema avatar-máscara

En la figura 7.1 se muestra la máscara colocada en la cabeza del robot. El casco que forma la cabeza del robot ha sido realizado en el Trabajo Fin de Grado concurrente con el actual al que ya se ha hecho referencia. Además, también se añadieron un conjunto de servomotores para el giro de la cabeza y el movimiento de cabeceo del cuello. Respecto a la máscara, esta va acoplada en la ranura de la pieza que rodea al rostro (en azul, en la figura 7.1). Por último, el proyector se ha situado en el cubículo en la parte trasera del casco, con una abertura para encender y apagar el proyector, y para conectar el cable HDMI.

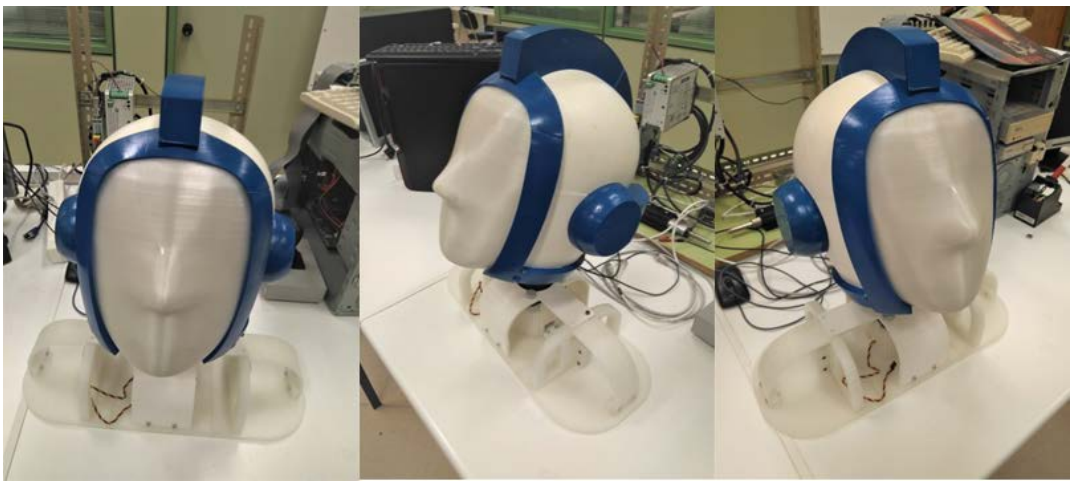


Figura 7.1 – Cabeza montada y máscara acoplada al casco

Con el fin de proyectar únicamente el rostro del avatar y no toda la cabeza en la máscara del robot, se va a emplear una "cámara". La cámara es un objeto de Blender que permite captar para video (animación), toda la escena hacia donde está orientada la propia cámara. Como solo se desea capturar el rostro, la cámara se va a situar en frente del avatar apuntando hacia el rostro. En el capítulo 6, se observó que el proyector tiene que estar rotado 90º para que la imagen se proyecte verticalmente sobre la forma de la máscara. Por consiguiente, la imagen que le llega al proyector también tiene que estar rotada 90º con el fin de que el avatar quede en posición vertical sobre la misma máscara. Esto se consigue girando la posición de la cámara en el editor 3D. En la figura 7.2, el lado de la cámara (la cámara es el rectángulo amarillo) con la solapa amarilla en forma de triángulo, constituye el lado superior de la imagen. En la figura 7.3 se muestra la imagen captada por la cámara.

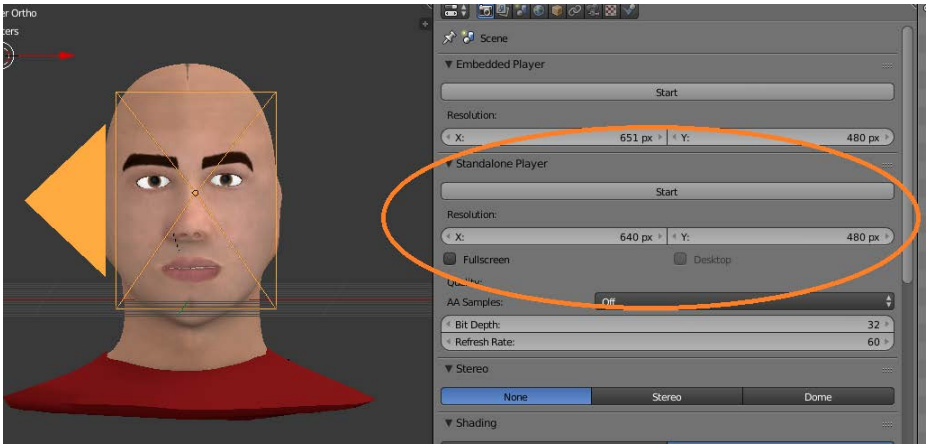


Figura 7.2 – Ajustes de posición y resolución de cámara en Blender

Blender además dispone de una característica en las propiedades de la cámara llamada *Standalone Player*. Esta propiedad permite ejecutar el motor de juego o la animación capturada por la cámara, en una ventana nueva.

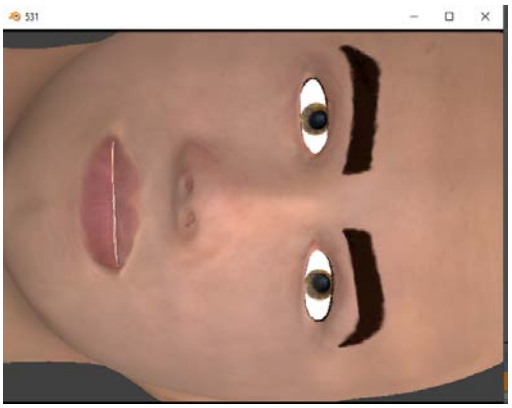


Figura 7.3 – Modo "Standalone Player"

Como el avatar se tiene que representar sobre la máscara, la opción *Standalone Player* resulta una opción muy práctica. Así se puede abrir el archivo .blend ejecutable como ventana externa sin tener que estar abriendo el programa Blender y estar cargando el archivo en pantalla.

Después se conecta el cable HDMI de la pantalla de trabajo al proyector. El último paso es configurar la resolución de imagen de la cámara de Blender para que la ventana encaje con el tamaño y silueta de la máscara. Esta resolución se ha ajustado a 640x480 pixeles (figura 7.2). El resultado se muestra en la figura 7.4.

Figura 7.4– Rostro del avatar proyectado sobre la máscara

La colocación del avatar es correcta ya que los puntos principales del rostro como la nariz, boca y labios se ajustan a la zona de la máscara que les corresponde, y el avatar queda centrado sobre la máscara. En cuanto al aspecto visual, resulta plenamente satisfactorio, con alguna pequeña irregularidad en la zona de la nariz derivada de la limitación del espesor de capa de la impresora 3D utilizada. En definitiva, el sistema en general ha quedado implementado de forma óptima. En el siguiente apartado, se van a mostrar los pasos para realizar la simulación con el gestor de comportamiento del avatar así como algunos ejemplos de las expresiones obtenidas.

7.3. Pruebas y resultado de las animaciones

7.3.1. Expresiones faciales

Antes de realizar la simulación sobre la cabeza robótica con el avatar proyectado en la máscara, se va a presentar el resultado de las principales expresiones faciales sobre el modelo avatar 3D. El avatar se mostrará tanto con la textura manual como con la textura obtenida a partir de la imagen de plantilla. En el apartado 3.3.2 se texturizó empleando ambos métodos. Junto a las imágenes del avatar, se va a mostrar también la imagen del autor para ver la comparación gestual entre las distintas expresiones. Aunque para el robot se ha empleado el modelo con textura manual, a continuación se van a mostrar ambos modelos ya que para el texturizado a partir de imagen de plantilla, se tomó como imagen una foto del autor. A pesar de crear dos texturas distintas, la malla base es la misma para ambas texturas, y por tanto la armadura de huesos y las unidades de acción siguen siendo las mismas. Es decir, a partir de un mismo modelo de malla 3D se pueden añadir distintas texturas manteniendo las propiedades del modelo, incluido huesos y formas clave ya que todos estos elementos están referenciados a la geometría de la malla 3D y no tienen relación con la textura que se le da a la superficie de la misma. A continuación se muestra un conjunto de ejemplos de expresiones faciales, incluyendo las seis expresiones básicas junto a las unidades de acción activas en cada caso.

Expresión: Neutra

UA activas: Ninguna

Figura 7.5 – Expresión: neutra

Expresión: Felicidad

UA activas: AU6 + AU12

Figura 7.6 – Expresión: felicidad

Expresión: Tristeza

UA activas: AU1 + AU4 + AU15

Figura 7.7 – Expresión: tristeza

Expresión: Miedo

UA activas: AU1 + AU2 + AU4 + AU5 +AU7 + AU20 + AU26

Figura 7.8 – Expresión: miedo

Expresión: Sorpresa

UA activas: AU1 + AU2 + AU5 + AU26

Figura 7.9 – Expresión: sorpresa

Expresión: Ira

UA activas: AU4 + AU5 + AU7+ AU23

Figura 7.10 – Expresión: ira

Expresión: Asco

UA activas: AU9 + AU20 + AU25

Figura 7.11 – Expresión: asco

Expresión: Enfado-llanto

UA activas: AU4 + AU7 + AU17

Figura 7.12 – Expresión: Enfado-llanto

Expresión: Desprecio

UA activas: AU4 + AU9 + AU10 + AU45

Figura 7.13 – Expresión: Desprecio

Para visualizar las animaciones del avatar sobre el rostro del robot, se ha realizado una simulación basada en la ejecución de un conjunto de rutinas de expresiones definidas en el gestor de comportamiento. El usuario puede modificar todos los parámetros gestuales del avatar. Los parámetros que se pueden modificar externamente a Blender son:

- *UA → Las unidades de acción que tendrá cada vector expresión facial (editar el diccionario de gestos, ver figura 4.6).*
- i_{UA} *→ La intensidad (de 0 a 1) asociada a cada unidad de acción en el vector expresión facial (editar en el diccionario de gestos).*
- $exp \rightarrow$ *Las expresiones faciales que ha de realizar el avatar (editar en el gestor de comportamiento, ver figura 4.4)*
- au_ext *→ Unidades de acción extra (editar en el gestor de comportamiento).*
- i_T *→ La intensidad total (de 0 a 1) de toda la expresión facial (editar en el gestor de comportamiento).*
- $v_{expresion}$ *→ Velocidad de cada expresión (editar en el gestor de comportamiento).*
- t_e *→ El tiempo de espera entre las expresiones (editar en el gestor de comportamiento).*

Existen otros parámetros que el usuario no puede manipular externamente, pero que se puede modificar en Blender. Estos parámetros son:

- $v_{animación}$ *→ La velocidad total de la animación en fotogramas por segundo.*
- $s_{unidades\_accion}$ *→ El espacio medido en fotogramas que abarca la unidad de acción, desde el fotograma inicial cuando la UA es nula, hasta el fotograma final, cuando la UA alcanza su intensidad máxima. En este caso todas las UA se han situado con el fotograma inicial en 0 y el fotograma final 10.*
- *Añadir si se desean más unidades de acción sobre el modelo 3D del avatar.*

El tiempo que dura cada expresión facial se calcula como:

$$t_{expresion} = \frac{s_{unidades\_accion}}{v_{animación} \times v_{expresion}} \quad (20)$$

La simulación se ha probado tanto en Ubuntu (ordenador) como en Raspbian con la Raspberry Pi 4. En ambos casos el sistema funciona, pero en la Raspberry Pi 4 la simulación se congela cuando se introduce una velocidad de animación alta. Esto se debe a que el procesador y memoria de la Raspberry es mucho más limitado que el ordenador. Sin embargo, a bajos fotogramas por segundo en la Raspberry se ejecuta correctamente. El inconveniente al emplear velocidades bajas, es que los movimientos de los gestos no son tan fluidos. La velocidad de animación recomendable para cada caso es:

- Ordenador: 12-24 fotogramas/segundo.
- Raspberry Pi 4: 1-2 fotogramas/segundo.

Para efectuar la simulación se abren distintas terminales para abrir primero el ejecutable de Blender y ejecutar luego cada uno de los nodos.

Orden de ejecución:

1º Compilar la carpeta de trabajo:
~/catkin_ws $ catkin_make

2º Se abre el ejecutable de Blender. Recibe datos de las unidades de acción a realizar, y se activan los actuadores correspondientes para mover el modelo avatar.
~/catkin_ws/src/modelo $ blenderplayer –w 854 480 0 0 avatar.blend

3º Abrir el nodo central:
~/catkin_ws/src $ roscore

4º Arrancar el archivo que conecta ROS con Blender. Recibe datos de las de las unidades de acción y se envían a Blender:
~/catkin_ws $ rosrun blender_control ros_blender_bridge.py

5º Arrancar el módulo de los gestos. Recibe datos de comportamiento, y envía datos de las unidades de acción al módulo puente.
~/catkin_ws/src/blender_control/src $ rosrun blender_control gestion_gestos.py

6º Arrancar el módulo de comportamiento. Lee el gestor de comportamiento, y envía datos de las expresiones a gestos.
~/catkin_ws/src/blender_control/src $ rosrun blender_control comportamiento.py comportamiento_avatar.yaml

Comienza la animación.

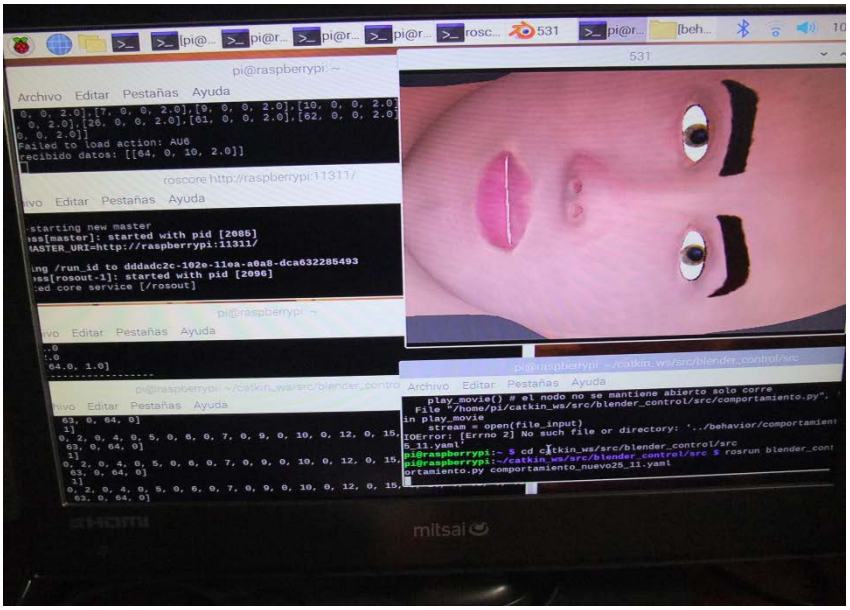


Figura 7.14 – Captura de imagen de la ejecución

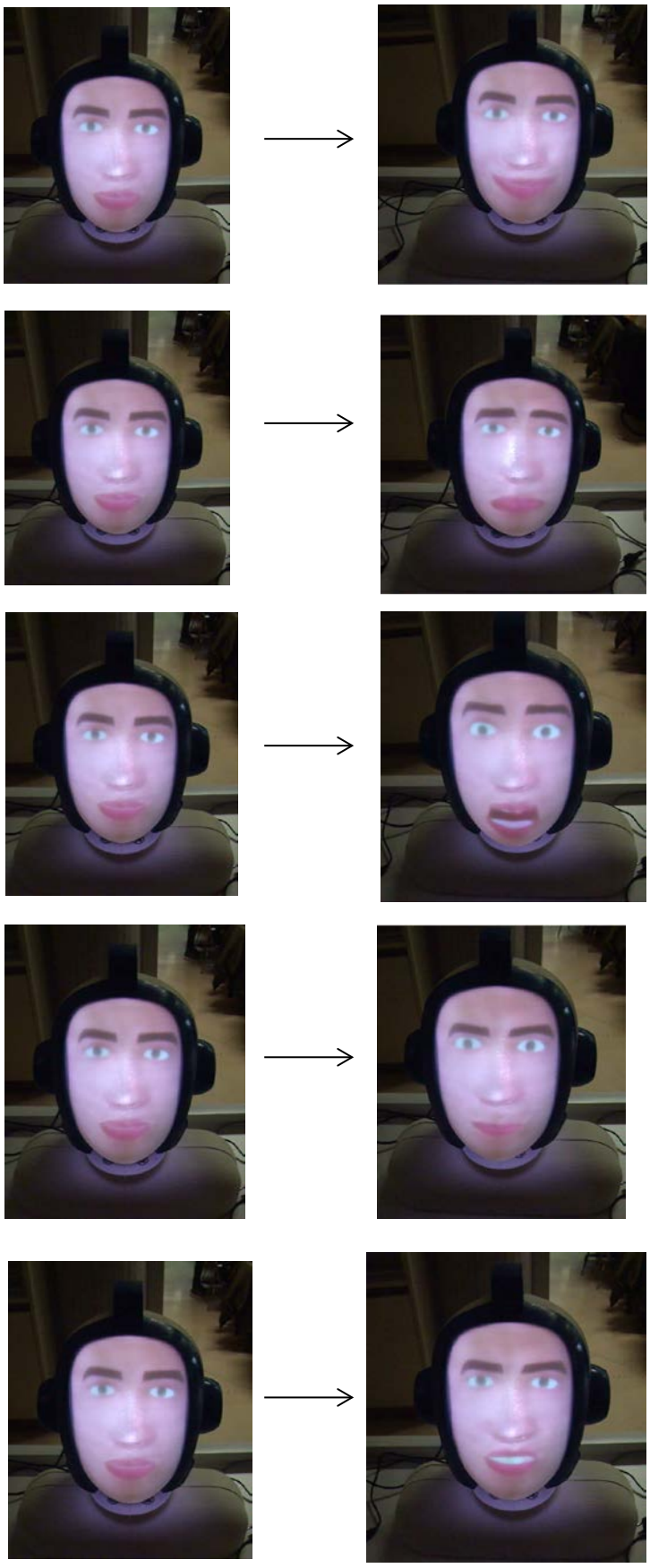
A continuación se muestra el archivo “comportamiento_avatar.yaml” con los datos de entrada, para esta simulación en específico. Seguidamente en la figura 7.16 se muestran capturas de las animaciones faciales de la simulación sobre el rostro del robot.

```
# ARCHIVO DE CONFIGURACION DE GESTOS.

esc_0:
  expresion:       neutra
  ua_extras:       []
  intensidad_total: 1
  velocidad:     0.25
  tiempo_espera:    4

esc_1:
  expresion:       alegria
  ua_extras:       []
  intensidad_total: 1
  velocidad:     0.25
  tiempo_espera:    4

esc_2:
  expresion:       neutra
  ua_extras:       []
  intensidad_total: 1
  velocidad:     0.25
  tiempo_espera:    4
```

```
esc_3:
  expresion:       tristeza
  ua_extras:       []
  intensidad_total: 1
  velocidad: 0.25
  tiempo_espera:    4

esc_4:
  expresion:       neutra
  ua_extras:       []
  intensidad_total: 1
  velocidad:     0.25
  tiempo_espera:    4

esc_5:
  expresion:       sorpresa
  ua_extras:       []
  intensidad_total: 1
  velocidad:     0.25
  tiempo_espera:    4

esc_6:
  expresion:       neutra
  ua_extras:       []
  intensidad_total: 1
  velocidad:     0.25
  tiempo_espera:    4

esc_7:
  expresion:       ira
  ua_extras:       []
  intensidad_total: 1
  velocidad:     0.25
  tiempo_espera:    4

esc_8:
  expresion:       neutra
  ua_extras:       []
  intensidad_total: 1
  velocidad:     0.25
  tiempo_espera:    4

esc_11:
  expresion:       desprecio
  ua_extras:       []
  intensidad_total: 1
  velocidad:     0.25
  tiempo_espera:    4
```

```
esc_12:
  expresion:       neutra
  ua_extras:       []
  intensidad_total: 1
  velocidad:     0.25
  tiempo_espera:    4

esc_13:
  expresion:       miedo
  ua_extras:       []
  intensidad_total: 1
  velocidad:     0.25
  tiempo_espera:    4

esc_14:
  expresion:       neutra
  ua_extras:       []
  intensidad_total: 1
  velocidad:     0.25
  tiempo_espera:    4

esc_15:
  expresion:       enfado
  ua_extras:       []
  intensidad_total: 1
  velocidad:     0.25
  tiempo_espera:    4

esc_16:
  expresion:       neutra
  ua_extras:       []
  intensidad_total: 1
  velocidad:     0.25
  tiempo_espera:    4
```

Figura 7.15- Rutina de comportamiento programada para este ejemplo



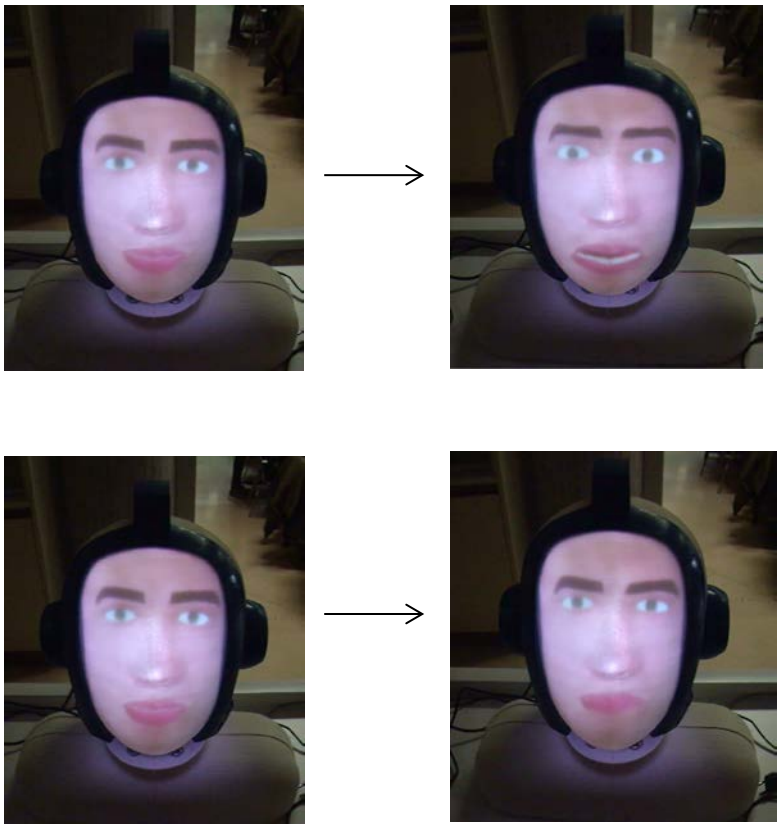


Figura 7.16-Ejecución del comportamiento del avatar sobre la cabeza. Izquierda: Posición neutra. Derecha: Expresión facial

En la figura 7.16 se muestran las distintas expresiones faciales del avatar proyectado sobre la cabeza siguiendo la rutina de comportamiento especificado. Entre expresión y expresión el rostro tiene que pasar a un estado neutro, ya que las formas clave de todas las unidades de acción están programadas en referencia a la forma base del modelo o malla. Del mismo modo, en la realidad el ser humano siempre pasa a un estado neutro entre expresiones, aunque dicha transición dure pocos milisegundos.

7.3.2. Voz y Sincronización labial en tiempo real

La simulación del habla y sincronización labial se puede realizar en cualquier sistema operativo donde se tenga instalado Blender y Python junto a las librerías necesarias (ver apartado 5.4). Para efectuar la simulación se abren distintas terminales, una para abrir primero el ejecutable de Blender y otra para ejecutar el módulo externo.

Orden de ejecución:

1º Se abre el ejecutable Blender del modelo. Previamente hay que ir al directorio donde esté ubicado el archivo del modelo.

```
>> blenderplayer –w 854 480  0 0 avatar1.blend
```

2º Ejecutar el archivo o módulo externo de la sincronización labial. Previamente hay que ir al directorio donde esté ubicado el archivo.

```
>> python lypsinc_external.py
```

Después de que se establezca la comunicación, el usuario escribe un texto de entrada en la terminal donde ha ejecutado el modulo externo. Finalmente comienza la animación en el ejecutable de Blender.

Figura 7.17- Ejecución de la sincronía labial con la voz en el avatar (en este caso el avatar con textura a partir de imagen)

En formato de imagen no se puede visualizar el funcionamiento de la voz y la sincronía labial del avatar. Es por eso que se ha realizado una captura de video de muestra, que se adjunta a la presente memoria, tal y como se explica en el anexo B. Después de ensayar el funcionamiento, se ha llegado a las siguientes conclusiones:

- Empleando los visemas de los fonemas consonánticos (/s/, /f/, /b/, /m/, /p/) junto a las vocales, la sincronía de la voz con los gestos no funciona correctamente. No hay una correspondencia simultánea de los visemas con sus correspondientes fonemas hablados ya que se produce un pequeño retardo en la gesticulación

respecto a la voz. Esto se debe a que el movimiento de todos los visemas se ha diseñado para que empiecen y finalicen en la posición neutra o de reposo de los labios. Se ha diseñado así porque si uno de los fonemas seleccionados se encuentra en la última posición del texto, la boca no puede quedar abierta. Esto a su vez causa un problema. Si se ejecutan dos visemas pertenecientes a fonemas seguidos (consonante y vocal normalmente) el efecto de la sincronía queda artificial ya que se tiene que pasar a la posición neutra entre fonemas consecutivos generando ese pequeño retardo, perdiendo así la fluidez y por ende perdiendo la sincronía con el habla. En contraposición, cuando el ser humano pronuncia una palabra en un texto no pasa a un estado neutro entre dos letras, sino que encadena de forma seguida los visemas consecutivos generando dicha fluidez en el habla.
Empleando en la animación solo los visemas de fonemas vocálicos (/a/, /e/, /i/, /o/, /u/) esta situación se corrige y la sincronía gestual con la voz funciona bastante mejor. Asimismo, aunque algunos visemas consonánticos son visibles al hablar, los visemas de las vocales son los más marcados por el ser humano durante el habla.

- Como se explicó en el apartado 5.4, la voz empleando el convertidor de Python “pyttsx3” resulta robótica y poco natural. El convertidor texto a voz gratuito con mejor voz es el de Google, aunque hasta el momento solo se le ha implementado voz femenina.

Capítulo 8 – Estudio económico

8.1. Introducción

Una vez finalizado el estudio técnico, en este capítulo se realizará un análisis del estudio económico del trabajo desarrollado. Hay que señalar que para este trabajo se ha partido de cero, por lo que no hay ningún coste previo al punto inicial del proyecto. El procedimiento a seguir va a ser elaborar los distintos costes y finalmente obtener el coste total. Los costes se van a dividir en costes directos e indirectos.

8.2. Costes directos

Los costes directos hacen referencia a los costes que guardan una relación estrecha con el producto o servicio realizado, en este caso el desarrollo del trabajo. Se ha considerado incluir como costes directos:

- Los costes de amortización de equipos y herramientas.
- Los costes de programas informáticos y/o sistemas operativos.
- Los costes de materiales consumibles.
- Los costes de personal.

Antes de calcular los costes se van a mostrar los días, horas laborables al año y el tiempo en horas empleado en realizar el presente trabajo. Estos datos se utilizarán posteriormente para el cálculo de algunos costes.

CÁLCULO DE LOS DÍAS LABORABLES AL AÑO	
CONCEPTO	**Días**
Días en un año	**365**
Días de vacaciones	**20**
Días de fines de semana	**104**
Días festivos	**14**
Total días laborables	**227**

Tabla 12 – Días laborables al año

$$Total\ horas\ laborables\ al\ año = 227 dias\ x \frac{8h}{dia} = 1816\ h \quad (21)$$

TIEMPO EMPLEADO EN LA REALIZACIÓN DEL PROYECTO	
CONCEPTO	**Tiempo (días)**
Total días (valor estimado)	**182**
	Tiempo (horas)
Total horas (6h/día)	**1092**

Tabla 13 - Tiempo empleado en el trabajo

- **COSTES DE AMORTIZACIÓN DE EQUIPOS Y HERRAMIENTAS**

Estos costes aluden a todos los elementos empleados en el proyecto que no se agotan, como equipos y herramientas. La amortización es el proceso por el cual un producto se devalúa en un periodo de tiempo resultado del desgaste producido por su uso o por el paso del tiempo. En este caso, se va a considerar una amortización lineal, donde el precio de los equipos disminuye en un valor constante por año. Las fórmulas empleadas para el cálculo de estos costes son:

El factor de amortización: Indica el valor de depreciación del equipo por año. Si la amortización es lineal este valor será el mismo para todos los años.

$$Factor\ de\ amortización\ \left(\frac{€}{año}\right) = \frac{Valor\ de\ compra - Valor\ residual}{N} \quad (22)$$

- *N: periodo de amortización en años.*
- *Valor de compra: Valor del producto en el momento de compra.*
- *Valor residual: Valor del producto al cabo del periodo de amortización o precio que se esperaría obtener por la venta del producto en el momento actual.*

$$Valor\ residual = Valor\ de\ compra \times (1 - Porcentaje\ estimado\ de\ devaluación) \quad (23)$$

El porcentaje de devaluación se expresa entre 0 y 1. Si el porcentaje de devaluación es del 5% (0,05), el valor residual será el 95% del valor de compra (0,95 × valor de compra).

Tasa horaria: Indica el coste de amortización por hora del equipo o herramienta.

$$Tasa\ horaria\ \left(\frac{€}{h}\right) = \frac{Factor\ de\ amortización}{horas\ efectivas/año} \quad (24)$$

Coste de amortización: Es el coste final del equipo calculado como el producto de su tasa horaria por las horas de uso en el proyecto.

$$Coste\ de\ amortización\ (€) = Tasa\ horaria\ \times horas\ de\ uso \quad (25)$$

Como la compra de todos los equipos, salvo el portátil, se ha realizado en el presente año, se va estimar un porcentaje de devaluación del 5% para cada uno de ellos, por el uso empleado durante este año. Para el portátil se va a estimar un porcentaje de devaluación del 40% respecto a su valor de compra, ya que este producto se compró hace siete años y su uso ha sido frecuente en todo este periodo.

COSTE DE AMORTIZACIÓN				
CONCEPTO	Valor de compra (€)	Valor residual (€)	N (años)	Factor amortización (€/año)
MiniProyector ExquizOn modelo S6	237	225,15	1	11,85
Lente tamaño móvil 180 ° ojo de pez	10	9,50	1	0,50
Adaptador cable microHDMI a HDMI marca UGREEN	9	8,55	1	0,45
Impresora 3D marca Ultimaker S3	3.372,50	3.203,87	1	168,63
Raspberry Pi 4 Modelo B	86,77	82,43	1	4,34
Portátil marca ASUS A55V i7 3610QM	760	456	7	43,43

Tabla 14 – Cálculo del factor de amortización de equipos

COSTE DE AMORTIZACIÓN				
CONCEPTO	Factor amortización (€/año)	Tasa horaria (€/h)	Horas de uso	Coste total (€)
MiniProyector ExquizOn modelo S6	11,85	0,0065	220	1,44
Lente tamaño móvil 180 º ojo de pez	0,50	0,0003	200	0,06
Adaptador cable microHDMI a HDMI marca UGREEN	0,45	0,0002	60	0,01
Impresora 3D marca Ultimaker S3	168,63	0,0929	15	1,39
Raspberry Pi 4 Modelo B	4,34	0,0024	60	0,14
Portátil marca ASUS A55V i7 3610QM	43,43	0,0239	700	16,74
TOTAL				19,78

Tabla 15 – Costes totales de amortización de equipos

- **COSTES DE PROGRAMAS Y SISTEMAS OPERATIVOS**

Costes procedentes de las herramientas informáticas empleadas en la realización del trabajo. Estos costes incluyen las licencias de los programas y sistemas operativos. Al igual que los equipos utilizados, estas herramientas informáticas no son consumibles, por lo que hay que calcular sus costes de amortización. El porcentaje de devaluación de la licencia del sistema operativo Windows 10 se ha estimado en un 20% (instalación hace tres años). El porcentaje de devaluación de la licencia del paquete Microsoft Office 2010 se ha estimado en un 60% (instalación hace siete años). El porcentaje de devaluación de la licencia del programa Photoshop CS6 se ha estimado en un 5% (instalación hace un año).

COSTE DE HERRAMIENTAS Y ENTORNO DE TRABAJO				
CONCEPTO	Valor de compra(€)	Valor residual(€)	N (años)	Factor amortización(€/año)
Licencias de sistemas operativos				
Sistema Operativo Windows 10 Home	145	116	3	9,67
Sistema Operativo Ubuntu 18.04 LTS	0	-	-	-
Raspbian para la Raspberry Pi 4	0	-	-	-
ROS versión "Melodic Morenia"	0	-	-	-
Licencias de programas				
Microsoft Office 2010 Home&Student	150	60	7	12,86
Adobe Photoshop CS6	270	256,50	1	13,50
Blender versión 2.79	0	-	-	-

Tabla 16 - Cálculo del factor de amortización de programas

COSTE DE HERRAMIENTAS Y ENTORNO DE TRABAJO				
CONCEPTO	Factor amortización(€/año)	Tasa horaria(€/h)	Horas uso	Coste total(€)
Licencias de sistemas operativos				
Sistema Operativo Windows 10 Home	9,67	0,0053	700	3,71
Sistema Operativo Ubuntu 18.04 LTS	0	-	-	0
Raspbian para la Raspberry Pi 4	0	-	-	0
ROS versión "Melodic Morenia"	0	-	-	0
Licencias de programas				
Microsoft Office 2010 Home&Student	12,86	0,0071	350	2,49
Adobe Photoshop CS6	13,50	0,0074	1	0
Blender versión 2.79	0	-	-	0
TOTAL				6,20

Tabla 17 – Costes totales de amortización de programas

- **COSTES DE MATERIALES CONSUMIBLES**

Los materiales consumibles son aquellos que se agotan con el proyecto. En la tabla 18 se muestran los costes.

COSTE DE MATERIAL	
CONCEPTO	**COSTE (€)**
Filamento PLA transparente 750-211399	**34,65**
Fotocopias y cartuchos tinta	**15**
TOTAL	**49,65**

Tabla 18 – Costes totales de material

- **COSTES DE PERSONAL**

El coste de personal se calculará como el coste de mano de obra, en este caso un ingeniero, para la realización del presente proyecto. El tiempo estimado que ha requerido el trabajo son 1092 horas (tabla 13). El número de días laborables al año son 227 días (tabla 12).

A continuación hay que tasar el sueldo de un ingeniero electrónico por hora. Se ha realizado una búsqueda para buscar el sueldo neto medio al año de estos profesionales en España con un nivel medio de experiencia. Se ha obtenido que el sueldo neto al año ronda los 25.000€. Este salario ya es el resultado de haber aplicado las retenciones fiscales al salario bruto. El cálculo del sueldo de un ingeniero por hora teniendo en cuenta la jornada laboral de 8h/día es:

$$Sueldo\,\frac{€}{dia} = \frac{25.000€}{227 dias} = 110{,}13\ €/dia \tag{26}$$

$$Sueldo\,\frac{€}{hora} = \frac{110{,}13€}{8\ horas} = 13{,}77\ €/hora \tag{27}$$

COSTE DE PERSONAL			
CONCEPTO	**Coste Unitario (€/h)**	**Duración(horas)**	**Costes total (€)**
Desarrollo del proyecto	**13,77**	**1092**	**15.036,84**

Tabla 19 – Coste de personal

Una vez conocido el coste total de personal, ya se puede obtener los costes totales directos como suma de los costes parciales.

COSTES DIRECTOS	
CONCEPTO	**COSTE (€)**
Coste de equipos y herramientas amortizables	19,78
Coste de materiales consumibles	49,65
Coste de programas informáticos	6,20
Coste de personal	15.036,84
TOTAL	**15.112,47**

Tabla 20 – Total de costes directos

Total costes directos = 15.112,47 €

8.3. Costes indirectos

Los costes indirectos son aquellos que no son directamente atribuibles a la obtención de un producto o servicio. En este caso, los costes indirectos son los producidos por la actividad requerida durante la elaboración del este trabajo. Se ha considerado incluir como costes indirectos el consumo de electricidad y desplazamiento. Para el cálculo del coste de consumo eléctrico se van a emplear las siguientes expresiones.

$$Precio\ consumo\ diario(\frac{€}{dia}) = Consumo\ diario\left(\frac{kWh}{dia}\right) \times precio\ de\ electricidad\ (\frac{€}{kWh}) \quad (28)$$

$$Precio\ consumo\ total\ (€) = Precio\ consumo\ diario(\frac{€}{dia}) \times número\ de\ dias\ de\ trabajo \quad (29)$$

Considerando el precio medio de la electricidad en España, el precio de electricidad se ha aproximado a un valor de $0{,}14\ €/\ kWh$.

CONSUMO ELÉCTRICO		
CONCEPTO	**Consumo diario (kWh/día)**	**Precio del consumo (€/día)**
Portátil marca ASUS A55V i7 3610QM	**0,520**	**0,073**
Raspberry Pi 4 Modelo B	**0,086**	**0,012**
TOTAL		**0,085**

Tabla 21 – Cálculo del precio de consumo eléctrico

$$Precio\ del\ consumo\ eléctrico\ total = 0{,}085\ \times 182 dias = 15{,}47€ \quad (30)$$

COSTES INDIRECTOS	
CONCEPTO	**COSTE (€)**
Consumo eléctrico de los equipos	**15,47**
Costes de desplazamiento	**12**
TOTAL	**27,47**

Tabla 22 – Total de costes indirectos

Total costes indirectos = 27,47 €

8.4. Coste total

El coste total resulta de la suma de los costes directos más los costes indirectos, y será el gasto total requerido en el presente proyecto.

COSTES TOTALES DEL PROYECTO	
CONCEPTO	**COSTE (€)**
Costes Directos	**15.112,47**
Costes Indirectos	**27,47**
TOTAL	**15.139,94**

Tabla 23 – Costes Totales

Coste total del proyecto = 15.139,94 €

Capítulo 9 – Conclusiones y líneas futuras

9.1. Conclusiones

En este apartado se van a exponer las conclusiones relativas al trabajo realizado. Estas conclusiones incluyen desde el grado la consecución del objetivo principal y los distintos sub-objetivos marcados al comienzo de la memoria (apartado 1.2), hasta la verificación de las características y ventajas de este tipo de rostros robóticos, también mencionadas en el capítulo 1 (apartado 1.1).

El presente TFG ha abordado el desarrollo de un avatar virtual 3D de aspecto humano y su integración en una cabeza robótica con el fin de crear un robot con un alto grado de expresividad. El rostro virtual desarrollado no solo presenta apariencia humana, sino que se ha logrado dotarle de un comportamiento gestual humano por medio de la combinación de distintas unidades de acción para obtener distintas expresiones faciales. Para la consecución de este resultado final, se han ido alcanzando cada uno de los sub-objetivos propuestos al comienzo:

- Desarrollo del avatar virtual empleando Blender. La ventaja de haber empleado este programa de modelado, además de que es multiplataforma, gratuito y libre, es que incluye el intérprete de Python, lo que ha permitido poder escribir y cargar scripts de este lenguaje para automatizar o controlar algunas tareas del modelo como se ha comprobado en apartados posteriores (control automático de las animaciones).
    - En la etapa de modelado, el modelo facial tridimensional se ha creado con ayuda de dos fotografías para obtener la apariencia y proporciones realistas de una cabeza humana.
    - En el texturizado se han empleado varias texturas para cada uno de los elementos del rostro. Para la piel se han creado dos texturas principales: una textura a partir de imágenes, y otra textura manual.
    - En cuanto a la animación, se ha logrado implementar un sistema de acción funcional, que permite generar distintos gestos y expresiones faciales, incluidas las principales expresiones emocionales del rostro humano. Este sistema se ha basado en la codificación FACS, formada por un conjunto de unidades de acción que simulan los movimientos faciales del ser humano. Asimismo, cada unidad de acción se ha compuesto de dos elementos: *Forma clave* o deformación de la malla tridimensional respecto a su forma base. Y el *hueso*, elemento encargado de controlar el valor de la

deformación. En el presente modelo ha incluido un total de 22 unidades de acción sobre el rostro, añadiendo la armadura con todos los huesos y sus correspondientes formas clave. Por último, cada una de estas unidades de acción se ha asociado a un actuador distinto en el editor lógico. Cuando el actuador se activa, se ejecuta la animación de la unidad de acción correspondiente, o en su caso, un conjunto de unidades de acción si se activan varios actuadores simultáneamente. Todo este sistema se ha englobado en el llamado nivel de control bajo, desarrollado en Blender.

- Respecto al control automático de las animaciones faciales, se ha logrado implementar un elevado nivel de control mediante ROS y Python. Esto ha permitido automatizar la ejecución de las distintas unidades de acción de forma externa a Blender, de forma que cuando se proyecta la imagen sobre la máscara acoplada al casco, dicha imagen corresponde a la ventana del motor de juego, sin necesidad de que la interfaz completa de Blender resulte visible. Este nivel ha constado de tres funciones principales:
  - La primera de las funciones es descomponer las expresiones faciales a representar en un vector de parámetros de unidades de acción.
  - La segunda función es traducir ese vector de unidades de acción, en una matriz legible por los actuadores de Blender.
  - La tercera y última función es la de enviar dicha matriz de parámetros a Blender. Finalmente el archivo Python cargado dentro de Blender, se encarga de recibir estos datos y activa automáticamente los actuadores asociados a las unidades de acción correspondientes.
  - La primera y segunda función se han llevado a cabo en distintos scripts de Python dentro de ROS, donde la comunicación entre nodos era mediante el método suscripción-publicación. La comunicación entre Python y Blender ha sido mediante sockets.

- En referencia al sistema avatar-máscara:
  - Se ha conseguido diseñar y fabricar la máscara translúcida que hace la función de pantalla de proyección. Esta máscara se ha modelado en Blender, con unas dimensiones similares a las de un rostro humano adulto. Después, se ha ajustado el contorno de dicha máscara a la forma de la ranura del casco. Para la fabricación, dicho modelo se ha enviado a una aplicación de la impresora 3D para generar el lenguaje de impresión y coordenadas. Por último se ha imprimido la pieza.
  - En cuanto al proyector, se ha elegido uno de dimensiones reducidas para poder ubicarlo dentro de la cabeza. Tras esto, se ha calculado la distancia de proyección, y se ha establecido un método de proyección óptimo que no comprometa o interfiera con la parte mecánica de la cabeza.

  - Una vez colocado el proyector en el cubículo dentro del casco, se ha acoplado la máscara impresa a la ranura realizada en la pieza del casco que rodea el rostro.

- Sistema de voz y sincronización labial en tiempo real: Por último se ha añadido esta nueva función al avatar. A partir de un texto de entrada, este sistema transforma el texto en habla empleando un convertidor texto a voz. De modo concurrente, el algoritmo diseñado también extrae los fonemas seleccionados del texto y estos se envían a Blender. Finalmente, el script Python cargado dentro de Blender se encarga de recibir los datos de los fonemas y ejecuta los actuadores de los visemas para generar la sincronía labial. Aunque este sistema pueda requerir de alguna mejoría, también sirve como un complemento de todo lo desarrollado previamente (sistema de generación-control de las animaciones faciales), lo cual hace que el modelo avatar sea más completo.

Tras el análisis de los objetivos y sub-objetivos llevados a cabo en el trabajo con resultados plenamente satisfactorios, también se pueden verificar los puntos comentados en el apartado 1.1, en referencia a las ventajas obtenidas con rostros robóticos basados en avatares.

- El primero de estos puntos citados eran los costes y componentes. En este caso, tanto los costes como los componentes han sido relativamente reducidos. Por un lado el programa de modelado Blender es gratuito. Por otra parte, el mayor gasto de los componentes proviene del proyector y la fabricación de la máscara (este gran gasto hace referencia a la adquisición de una impresora 3D si no se dispone de una. No obstante, el coste de amortización de la impresora para este trabajo es muy pequeño).
- En cuanto a mantenimiento, este sistema no requiere de una supervisión continua del estado de los componentes. Basta con mantener la máscara, el proyector y la lente en un ambiente seco y a una temperatura adecuada.
- Otro de los puntos citados, hacía referencia a la flexibilidad de edición y reutilización que ofrecen los modelos computacionales. Una vez desarrollado el modelo, se puede verificar esta flexibilidad ante cambios en la apariencia del avatar. Aunque para el modelado del avatar se hayan empleado dos fotos del autor como base, el formato de malla final es el de una cabeza genérica (con fisionomía masculina). Después a mayores se añadieron dos texturas distintas sobre esta malla para dar distinta apariencia al rostro. Esto demuestra que en el futuro se puede reutilizar esta misma malla 3D de forma directa si se desean añadir otras texturas para generar distintos rostros.
- Comportamiento gestual y expresividad: El sistema empleado está basado en la codificación FACS, pero además a cada unidad de acción se le ha asignado unos parámetros: intensidad y tiempo de ejecución. También se ha incluido un nivel de

intensidad total para la expresión facial generada, independientemente del nivel de intensidad que tenga cada unidad de acción por separado. De este modo, se obtiene mayor complejidad y variedad gestual sobre el rostro del modelo. Ya que para dos expresiones faciales con las mismas unidades de acción activas, una de ellas puede denotar mayor fuerza o emoción que otra, en función del grado de intensidad total de la expresión.

En definitiva, el presente trabajo ha permitido confirmar que el desarrollo de cabezas robóticas en combinación con tecnologías de modelado y diseño de avatares virtuales, representa una alternativa idónea a robots sociales basados en rostros de naturaleza electromecánica, ofreciendo gran flexibilidad en cambios de apariencia y variedad gestual de comportamiento.

9.2. Líneas futuras

En cuanto a líneas futuras, este proyecto ofrece posibilidades de avance. Al igual que ocurrió con los primeros modelos de otros robots como el *Mask-bot* o *Furhat*, se puede avanzar en conceptos nuevos para obtener posteriores modelos. A continuación se van a evaluar algunas líneas futuras que se pueden seguir a partir del estado actual.

Uno de los posibles avances puede ser el controlar el movimiento y las expresiones del avatar a partir de eventos externos que capte el robot del entorno, en vez de emplear un gestor de comportamiento. Los “actuadores” del avatar, que serían las unidades de acción, ya están implementados en el modelo actual que se ha desarrollado. Así pues, bastaría añadir un sistema de sensores y control, de modo que al detectar un cierto patrón externo, se active el movimiento de dichos actuadores faciales. Un ejemplo sería añadir una cámara en la cabeza del robot y mediante sistemas de reconocimiento facial, detectar a individuos en frente del robot. Si estos individuos se desplazan fuera del campo de visión de la cámara, el robot podría girar el cuello y a la vez también girar los ojos en coordinación con el cuello. Es decir que se active la unidad de acción AU61 o AU62 del avatar según la dirección hacia donde gire el cuello. Actualmente, se cuenta con un investigador postdoctoral que está desarrollando su labor en esta línea de trabajo.

Otra de las posibles líneas que se puede seguir es en el ámbito de la comunicación oral. En el modelo actual se ha implementado un sistema de voz con sincronía labial en tiempo real, a partir de la conversión texto a voz de un texto de entrada. A partir de aquí, la línea futura puede consistir en añadir un sistema de conversión voz a texto de forma que a su vez se pueda analizar la frase/texto captado tras la conversión y en función del contenido semántico, el robot pueda decidir la respuesta que luego se llevaría al avatar. Esta

respuesta en forma de texto pasaría finalmente por el convertidor texto a voz desarrollado, generando así la respuesta hablada del avatar a una entrada también de voz. De este modo se obtendría una comunicación verbal bidireccional entre el hombre y la máquina. Quizá la parte más compleja de este sistema voz a texto, sea el reconocimiento de habla para que la transformación voz a texto sea fidedigna.

Por último, respecto al sistema de sincronización y voz desarrollado en este trabajo, aunque el funcionamiento sea satisfactorio, también se puede buscar algún otro método que permita añadir algunos fonemas consonánticos a mayores junto a las vocales, sin que se generen retardos y que mantenga la sincronización labial con el habla.

Referencias Bibliográficas

Las referencias bibliográficas están redactadas acorde a la normativa IEEE.

[1] ISEA S.COOP, «Internet 3D, Análisis prospectivo de las potenciales aplicaciones asociadas a los mundos virtuales.,» Octubre 2008. [En línea]. Available: http://www.iseamcc.net/eISEA/Vigilancia_tecnologica/informe_3.pdf. [Último acceso: 2019 Noviembre 7].

[2] Indra, Fundación Universia y U-tad., «Proyecto CicerOn:Cuando la realidad virtual ayuda a la capacitación de las personas con síndrome de Asperger,» 18 Julio 2016. [En línea]. Available: https://www.indracompany.com/sites/default/files/indra_proyecto_ciceron_u-tad_f_universia.pdf. [Último acceso: 7 Noviembre 2019].

[3] L. M. Dafa, C.Baudracco, A. Moragas, C.L. Vila, D. M. Trujillo y P.V. Estrada, «KRISTINA, un asistente virtual para superar las barreras lingüísticas en la,» 2017. [En línea]. Available: https://repositori.upf.edu/bitstream/handle/10230/34244/dafa_comunidad19_KRIS.pdf?sequence=1&isAllowed=y. [Último acceso: 2019 Noviembre 7].

[4] «KRISTINA,» [En línea]. Available: http://kristina-project.eu/en/. [Último acceso: 7 Noviembre 2019].

[5] J.J. Chini, C.L. Straub and K.H. Thomas, «Learning from avatars: Learning assistants practice physics pedagogy,» *Physical Review Physics Education Research,* vol. 12, 2016.

[6] T.Kuratate, B.Pierce and G.Cheng, «"Mask-bot": A life-size talking head animated robot for AV speech and human-robot communication research,» *AVSP,* pp. 111-115, 2011.

[7] B.Pierce,T.Kuratate, C.Vogl and G.Cheng, «"Mask-Bot 2i": An active customisable Robotic Head with Interchangeable Face,» *IEEE-RAS International Conference on Humanoid Robots,* pp. 520-525, 2012.

[8] Furhat Robotics, «Furhat Robot,» [En línea]. Available: https://www.furhatrobotics.com/. [Último acceso: 9 Noviembre 2019].

[9] S. Al Moubayed, J.Beskow, G.Skantze and B.Granström, «Furhat: A Back-Projected Human-Like Robot Head for Multiparty Human-Machine Interaction,» 2012.

[10] «Avatar SDK,» [En línea]. Available: https://webdemo.avatarsdk.com/. [Último acceso: 9 Noviembre 2019].

[11] K.L Novak and C.Rauh, «tChoose your ‘buddy icon’ carefully: The influence of avatar androgyny, anthropomorphism and credibility in online interactions.,» *Computers in Human*

Behavior, vol. 24, pp. 1473-1493, 2008.

[12] H.Brenton, M.Gillies, D.Ballin and D.Chatting, «The Uncanny Valley: does it exists?,» 2005.

[13] C.Ho, K.MacDorman and Z. Pramono , «Human emotion and the uncanny valley: A GLM, MDS, and ISOMAP analysis of robot video ratings,» *Proceedings of the 3rd ACM/IEEE International Conference on Human Robot Interaction,* p. 175, 2008.

[14] C.S.Burgos y I.R.Blanco, «Modelado y animación facial de un avatar virtual realista,» Universidad de Valladolid, 2013, p. 46.

[15] P. Ekman and W.V. Friesen, «Constants across cultures in the face and emotion,» *Journal of Personality and Social Psychology,* vol. 17, nº 2, pp. 124-128, 1971.

[16] R.Jack, O.Garrod, H.Yu, R.Caldara and P.Schyns, «Facial expressions of emotion are not culturally universal,» *Proceedings of the National Academy of Sciences of the United States of America,* vol. 109, pp. 1-4, 2014.

[17] C.S.Burgos y I.R.Blanco, «Modelado y animación facial de un avatar virtual realista,» Universidad de Valladolid, 2013, pp. 49-50.

[18] G. Duchenne, «The Mechanism of Human Facial Expression,» Cambridge University Press, 1990, pp. 275-278.

[19] P.Ekman and W.V.Friesen, «Smiles when lying,» *Journal of Personality and Social Psychology,* vol. 54, nº 3, pp. 414-420, 1988.

[20] Fundación Blender, «Blender Reference Manual,» [En línea]. Available: https://docs.blender.org/manual/en/latest/. [Último acceso: 21 Octubre 2019].

[21] M.Núñez, F. León, P. F. Cardenas, «ROS sistema operativo para robótica, nociones y aplicaciones,» *Revista Colombiana de Tecnologías de Avnzada,* pp. 1-6, 2014.

[22] W.C.Pang, Burhan, G.Sheet, «Design considerations of a robotic head for telepresence applications,» p. 6, 2012.

[23] D.Loza, S.M.Pablos, E.Z.Casanova, J.G.Bermejo, «Animation of Expressions in a Mechatronic Head,» de *First Iberian Robotics Conference*, 2013.

[24] D.Loza, S.Marcos, E.Z.Casanova, J.G.Bermejo, «Interaction system based on an avatar projected on a pyramidal display,» de *International Conference on Intelligent Robots and Systems*, Madrid, 2018.

[25] D.Loza, S.Marcos, E.Zalama, J.G.Garcia-Bermejo, J.L.Gonzalez, «Application of the FACS in the Design and Construction of a Mechatronic Head with Realistic Appearance,» *Journal of physical agents,* vol. 7, nº 1, p. 36, 2013.

[26] «Facial Action Coding System,» [En línea]. Available: https://en.wikipedia.org/wiki/Facial_Action_Coding_System. [Último acceso: 12 Noviembre 2019].

Anexos

En el apartado de Anexos está incluida cierta información que por extensión o especificación no se ha considerado integrar dentro de los apartados regulares de la memoria. Los anexos son los siguientes:

-Anexo A: Tablas de comandos empleados en Blender.

-Anexo B: Organización de los directorios, asociados al trabajo y ejecución de resultados.

-Anexo A

En este anexo se van a mostrar los comandos de Blender más importantes, y los que han sido de utilidad para el diseño del avatar y las animaciones. Los comandos sirven en muchos casos como atajo a la hora del manejo de las herramientas y operaciones de trabajo. Si no se ha manejado previamente Blender o no se tiene un conocimiento previo, estas tablas de comandos son de gran utilidad.

COMANDO	ACCIÓN
1	Vista de alzado(frente)
3	Vista perfil derecho
Cntrl + 3	Vista perfil izquierdo
5	Cambio entre vista en modo perspectiva y modo ortogonal
7	Vista de planta (desde arriba)
Cntrl + 7	Vista de planta (desde abajo)
2,4,6,8	Rotación del area de trabajo
0	Vista de la cámara
.	Centrar la vista en el objeto seleccionado
/	Desaparecen el resto de objetos de la escena dejando solo a la vista el objeto seleccionado

COMANDO	ACCIÓN
G	Traslación del objeto
G+X	Traslación en el eje X
G+Y	Traslación en el eje Y
G+Z	Traslación en el eje Z
R	Rotación del objeto
R+X	Rotación en el eje X
R+Y	Rotación en el eje Y
R+Z	Rotación en el eje Z
S	Escalado del objeto
S+X	Escalado en el eje X
S+Y	Escalado en el eje Y
S+Z	Escalado en el eje Z
N	Se abre panel de transformación
E	Extrusión de vértices
E+T	Extrusión más traslación
E+R	Extrusión más rotación
E+S	Extrusión más escalado

COMANDO	ACCIÓN
BDR	Seleccionar objeto (modo objeto)□Seleccionar vértice (modo edición)
BIR	Mover el puntero
A	Seleccionar/Deseleccionar todos los objetos (modo objeto) □ Seleccionar/Deseleccionar todos los vertices(modo edición)
B+BIR	Seleccionar vértices dentro de un area cuadrada
C+BIR	Seleccionar vértices dentro de un area circular
Tab	Cambiar de modo objeto a modo edicion
Cntrl+R	Subdivisión de las caras,mediante la creación de un loop edge
X	Borrar objeto seleccionado
Shift + A	Añadir elemento a la escena
Alt + D	Duplicar objeto
Cntrl +E	Herramientas de bordes
Cntrl+J	Unir dos objetos en un mismo mesh (malla)
P	Separar mesh en dos objetos distintos
F	Unión de los vértices seleccionados, mediante aristas
Cntrl + BIR	Crear nuevos vértices en un mesh
M	Cambiar de layer(capa) el objeto seleccionado
U	Desenvoltura de la malla 3D en mapa 2D (unwrap)
Cntrl + D	Añadir driver a la propiedad de un objeto
Cntrl+P	Emparentar un objeto o hueso a otro
Alt + P	Eliminar emparentamiento
I	Insertar fotograma clave(keyframe)
Alt + A	Iniciar reproducción de la animación

Tabla 24 – Comandos de uso de Blender

-Anexo B

En un principio se iba a plasmar en este anexo el código de los archivos Python asociados al control de las expresiones del avatar y sincronización de voz. Sin embargo por extensión se ha decidido no incluirlos. Asimismo en los apartados correspondientes dentro de la memoria, ya se ha ido explicando en profundidad el funcionamiento de cada uno de estos archivos, por lo que no se ha considerado incluir el código de nuevo. En este anexo se va a mostrar la organización y el contenido del trabajo que se entregará junto a esta memoria. Los contenidos a entregar y directorios son los siguientes:

-Memoria del trabajo

-Archivos Blender del modelo del avatar.

-Expresiones faciales (Directorio)

- Carpeta catkin_ws: En esta carpeta esta contenido todo los nodos de ROS empleados para el control de las expresiones faciales.
  - ros_blender_bridge.py
  - gestión_gestos.py
  - comportamiento.py
  - diccionario_gestos.yaml
  - comportamiento_avatar.yaml
  - ua_avatar_blender.py

-Sincronización de voz (Directorio)

- Lypsinc_external.py
- Lypsinc_blender.py
- Video de muestra

Por último comentar que en el presente proyecto se ha contado con la ayuda presencial del profesor David Loza para la implementación del nivel alto de control de las expresiones faciales. David ha realizado varios proyectos previos relacionados con esta temática, acerca de la animación gestual de avatares [23] [24]. Los archivos de control se han aplicado al presente trabajo, con variaciones en las unidades de acción, gestor de comportamiento, y diccionario de expresiones.