Universidad Zaragoza
1542

Trabajo Fin de Máster

Sistema de realidad virtual para exploración 3D con visión protésica simulada

Virtual reality system for 3D navigation with simulated prosthetic vision

Autora

María Santos Villafranca

Directores

Jesús Bermúdez Cameo

Alejandro Pérez Yus

Escuela de Ingeniería y Arquitectura

2021

Sistema de realidad virtual para exploración 3D con visión protésica simulada

RESUMEN

Las personas dependen de los sentidos para navegar por el entorno e interactuar con él y, sin duda, el sentido en el que más se confía es la visión, ya que la mayor parte de la información recibida es a través de él. Sin embargo, existe un gran número de personas privadas de este sentido, ya sea de nacimiento o por alguna patología o enfermedad degenerativa que pueden llegar a causar ceguera total o parcial. Las prótesis visuales tratan de mejorar la calidad de vida de estas personas. Existen diferentes tipos de prótesis que se pueden situar en la retina, la corteza visual o el nervio óptico en función del problema que cause la ceguera. La más usada mundialmente es el Argus II que consta de una microcámara que capta la información visual que posteriormente es convertida a estimulaciones eléctricas lo que permite ver puntos de luz denominados fosfenos. Desgraciadamente, estos dispositivos poseen una resolución muy reducida y su campo de visión muy limitado, en torno a los 20º, por lo que se están investigando diferentes representaciones de mapas de fosfenos para mostrar más información del entorno con dichas limitaciones y mejorar así la interacción de los pacientes con el entorno.

En este proyecto se ha desarrollado un simulador de visión fosfénica que permite, mediante las gafas de realidad virtual Oculus Rift DK2 conectadas desde un cliente de realidad virtual con visión protésica, y el simulador Gazebo ejecutado desde otro servidor Ubuntu en remoto, mostrar un entorno virtual en tiempo real suficientemente realista, por el cual se puede navegar libremente y explorar por completo moviendo la cabeza. Además, este simulador permite visualizar diferentes tipos de representaciones de mapas de fosfenos para poder realizar experimentos en un futuro y se ha introducido una red neuronal pre-entrenada para poder detectar diferentes objetos de la escena y resaltarlos para facilitar el reconocimiento de los mismos.

El código se ha implementado en leguaje C++ y python con la propia interfaz de programación de aplicaciones (API) de Oculus, las librerías OpenCV para el manejo de imágenes, el sistema operativo ROS y el simulador Gazebo.

Virtual reality system for 3D navigation with simulated prosthetic vision

ABSTRACT

People depend on their senses to navigate the environment and interact with it and, without a doubt, the most trusted sense is vision, since most of the information received is through it. However, there is a large number of people deprived of this sense, either by birth or by some pathology or degenerative disease that can cause total or partial blindness. Visual prostheses try to improve the quality of life of these people. There are different types of prostheses that can be placed in the retina, the visual cortex or the optic nerve depending on the problem causing blindness. The most widely used worldwide is the Argus II, which consists of a microcamera that captures visual information which is then converted to electrical stimulation, allowing the patient to see points of light called phosphenes. Unfortunately, these devices have a very low resolution and a very limited field of view, around 20º, so different representations of phosphene maps are being investigated to show more information of the environment with these limitations and thus improve the interaction of patients with the environment.

In this project we have developed a phosphenic vision simulator that allows, through virtual reality goggles Oculus Rift DK2 connected from a virtual reality client with prosthetic vision, and the Gazebo simulator running from another Ubuntu server in remote, display a sufficiently realistic real-time virtual environment, which can be freely navigated and fully explored by moving the head. In addition, this simulator allows to visualize different types of phosphene map representations for future experiments and a pre-trained neural network has been introduced to detect different objects in the scene and highlight them for easier object recognition.

The code has been implemented in C++ and Python languages with Oculus' own application programming interface (API), the OpenCV libraries for image management, the ROS operating system and the Gazebo simulator.

Índice

Índice de Figuras

Índice de Tablas

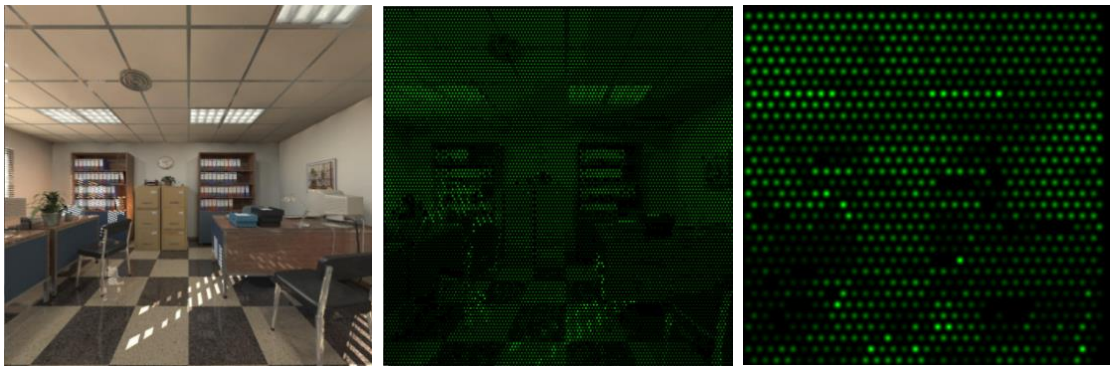
Capítulo 1

Introducción

La vista es el sentido que más información proporciona, y por el que las personas más se guían. Sin embargo, no todas las lo poseen. Según la Organización Mundial de la Salud (OMS), aproximadamente 1300 millones de personas viven con alguna deficiencia visual, de las cuales 36 millones son ciegas [1], y es por eso que en los últimos años la investigación en visión artificial para asistencia a personas con discapacidad visual es tan importante.

Dentro de los muchos ámbitos que abarca la asistencia a personas con discapacidad visual, este proyecto se centra en el de las prótesis visuales, que son aquellas capaces de estimular diferentes partes de la vía visual, según el problema que cause la ceguera, y percibir puntos luminosos llamados *fosfenos.*

Las prótesis visuales intentan mejorar la calidad de vida de personas que han perdido total o parcialmente la visión debido a enfermedades degenerativas, como la retinitis pigmentosa o la degeneración macular. En la primera, los fotorreceptores mueren y en la segunda se degenera la parte central de la retina, sin embargo, el circuito retinal restante no, lo que permite que se pueda estimular mediante impulsos eléctricos y que se generen puntos luminosos llamados *fosfenos*. En la Figura 1.1 muestra un ejemplo de cómo vería una persona con prótesis visual:



(a) **(b)** **(c)**

Figura 1.1. Mapa de fosfenos de una habitación [2]

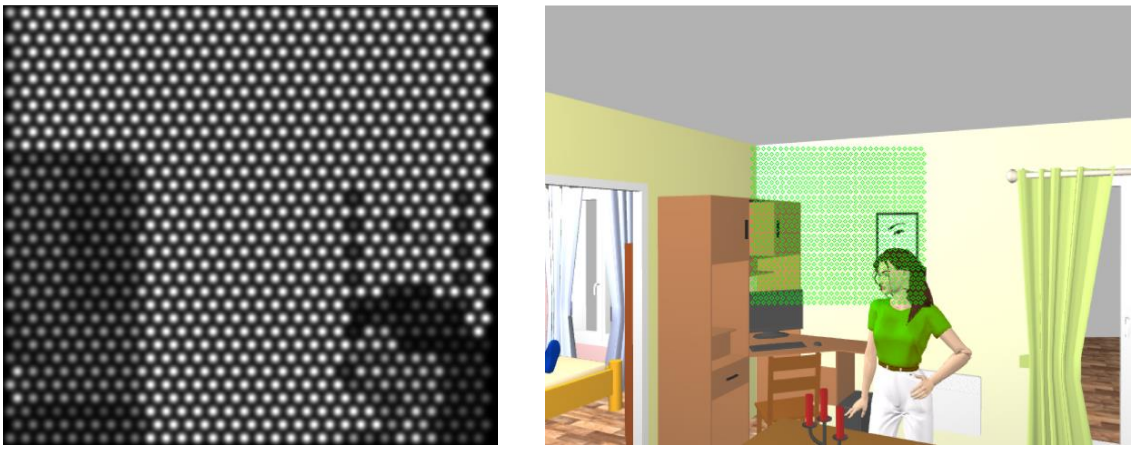
En la Figura 1.1.a. se muestra la imagen original, en la Figura 1.1.b. la misma imagen en mapa de fosfenos con 10.000 fosfenos en el visor de RV y en la Figura 1.1.c. con 1.000 fosfenos. Se puede apreciar la importancia del número de fosfenos para poder interpretar el entorno u objetos del mismo.

En este proyecto, las imágenes han sido capturadas mediante una cámara virtual en un simulador, donde, además de capturar imágenes desde el punto de vista de la cámara virtual, se puede monitorizar el movimiento de la persona por dentro del escenario del entorno virtual desde una vista externa, tal y como se muestra en la Figura 1.2:

Figura 1.2. Entorno virtual simulado

A pesar de que este simulador no es completamente fotorealista, se considera que sí que alcanza un nivel de realismo suficiente para simular prótesis visuales, ya que una parte importante de los detalles de textura e iluminación se pierden en la reducción de resolución espacial y rango dinámico que se da al generar la imagen de fosfenos. Si se compara el resultado (Figura 1.3.) con una fotografía real (Figura 1.1) se puede apreciar

el parecido entre las imágenes de fosfenos de ambos casos a pesar de que el nivel de fotorealismo de las imágenes originales es muy distinto:



a) 1.000 fosfenos con 20º de campo de visión **b) Imagen original**

Figura 1.3. Comparación de visión de persona con prótesis a persona con visión normal

En la Figura 1.3.a se puede apreciar lo que vería una persona con un implante, mientras que en la Figura 1.3.b se muestra la imagen al completo que vería una persona con visión normal, donde la zona verde es el campo de vista que visualiza la persona operada, en este caso de 20º, que es el máximo campo de visión que las prótesis visuales son capaces de ofrecer actualmente.

Las personas con estas prótesis disponen de pocos fosfenos y un campo de visión bastante reducido, por lo que cualquier prueba experimental resulta muy complicada, además existen muy pocas personas en el mundo operadas. Por ello, tal y como explica el artículo [3], se han generado los simuladores de visión protésica (SPV) para poder realizar experimentos no invasivos en sujetos con visión normal e implementar técnicas que ayuden a interpretar mejor el entorno.

A priori, parece que la solución más obvia sería intentar representar lo que ve una cámara (similar a lo que ven unos ojos) trasladando la información directamente al mapa de fosfenos. Pero la información que se puede mostrar con una prótesis visual es muy limitada, además de resolución muy baja (pocos fosfenos), rango dinámico muy bajo (pocos niveles de intensidad), poco campo de vista y otros problemas como ruido. Por lo tanto, es necesario investigar otros modos de trasladar la información más relevante de la escena a algo que pueda entenderse en un sistema tan limitado.

1.1. Motivación

La carencia de visión resulta especialmente frustrante para aquellas personas que han podido ver en el pasado. Éste es el caso de personas ciegas no congénitas, que han perdido la vista total o parcialmente debido a enfermedades degenerativas o accidentes.

Los avances en este campo de investigación pueden ayudar a estas personas reparar esa carencia. Los pacientes en los que actualmente se centra esta investigación son aquellos con enfermedades degenerativas como la retinitis pigmentosa, coroideremia o degeneración macular. Desde los años 60, varios trabajos han comprobado que la estimulación eléctrica de la vía visual permite generar puntos de luz llamados fosfenos [4]. Esta estimulación puede aplicarse en la retina, el nervio óptico o en la corteza visual [5] y generan un mapa de fosfenos similar a una imagen de puntos de baja resolución como se muestra en la Figura 1.3.a. o en la Figura 1.1.b. y Figura 1.1.c. Por desgracia, la resolución de la rejilla de fosfenos producidos se ve limitada por la biología, la tecnología y la seguridad del propio paciente [6]. Los dispositivos actuales proporcionan entre 60 y 1.000 fosfenos.

Actualmente, existen numerosos trabajos centrados en mejorar la percepción del entorno, ayudar la navegación por el mismo, como el reconocimiento de personas, objetos o estancias. Sin embargo, no existe ningún simulador capaz de permitir una experiencia inmersiva y a la vez permitir el movimiento total por un entorno virtual, que permita explorar estancias u objetos con la posibilidad de examinarlos desde todas las posiciones posibles. Un entorno de simulación permite la realización sistemática de experimentos masivos en igualdad de condiciones obteniendo conclusiones con sentido estadístico y monitorizando el experimento de forma segura. Estás condiciones pueden ser modificadas de forma muy flexible adaptando el escenario o las diferentes representaciones de fosfenos a las necesidades del experimento. Además, facilita tareas que serían muy complicadas en un entorno real como son la recogida de datos automática y la localización espacial del sujeto en todo momento.

1.2. Estado del Arte

Actualmente el número de pacientes operados con prótesis es muy bajo y por tanto no resulta viable experimentar con ellos. Por ello los SPV son la técnica más prometedora

para evaluar la forma en que se comunica la información visual a los pacientes, además, se pueden estudiar los requisitos mínimos para realizar diferentes tareas y analizar hacia donde debe avanzar la investigación.

Uno de los campos en los que se centra esta investigación es en la navegación segura y eficaz por entornos, por ejemplo Cha en [7] encontró que con un campo de visión de 30º e introduciendo una serie de píxeles de 25 × 25 distribuidos dentro del área visual foveal podría proporcionar una movilidad útil en entornos que no requieren un alto grado de reconocimiento de patrones.

Rheede en [8] propuso un SPV con dispositivo montado en la cabeza con seguidor ocular, que, al igual que en este trabajo, una de las ventajas es evaluar cómo el escaneo visual con movimientos de la cabeza mejora la agudeza visual [9].

Debido a la gran variabilidad existente a la hora de diseñar SPVs, Chen en [10], [11] propone un marco estandarizado para que las simulaciones futuras puedan configurarse para ser más realistas comparando las representaciones del simulador con la descripción obtenida de sujetos que han probado la prótesis. Por otra parte, también se están usando técnicas de visión por computador y deep learning [12] para intentar representar el máximo de información en los fosfenos disponibles.

Dentro de la propia Universidad de Zaragoza se han creado diferentes SPV, como es el caso de [13] que captura la información de la escena mediante la cámara Kinect for Windows que introduce información de profundidad. En [14] se plantea la integración de un sistema de guiado de visión protésica basado en técnicas de navegación convencionales. En [2] se crea un entorno virtual a partir de imágenes panorámicas que permite ser explorado gracias a las Oculus Rift DK2.

Existen otros trabajos dentro de este campo de investigación como, por ejemplo para el reconocimiento de texto [15] o de personas [16], pero los que más interesan para este proyecto son aquellos que utilizan realidad virtual. Entornos que son usados para evaluar la respuesta del usuario con diferentes modelos de representación visual [17], [18], [19], [20], [21], [22]. Dagnelie en [17] exploró los requisitos mínimos de resolución visual de un conjunto de electrodos simulados para la movilidad en entornos reales y virtuales. Se requirieron 6x10 puntos con un campo de visión de 27ºx16.2º para un desempeño adecuado. Zhao en [22] simuló un entorno visual más complejo con personas

de visión normal como sujetos de prueba y les encomendó tareas de identificación y búsqueda de caminos con 50º de campo de visión.

En este proyecto se va a utilizar un SPV inmersivo basado en un entorno virtual ya que este tipo de entornos permite probar nuevas representaciones y realizar experimentos con personas de manera realista, pero reduciendo la complejidad del experimento, en una gran variedad de escenarios de forma segura y evaluar el impacto del número de fosfenos, el tamaño, representaciones, etc.

A diferencia de otros SPV existentes, este está basado en un framework de robótica, lo cual permite trabajar con múltiples sensores realistas que replican sensores existentes y por tanto permitirían transportar los resultados al mundo real muy rápidamente. Además, se pueden incorporar multitud de métodos ya desarrollados en el ámbito de la robótica para añadir nuevas funcionalidades (navegación, SLAM, tratamiento de nubes de puntos).

En este trabajo se va a dotar al sistema de capacidades inmersivas mediante gafas de realidad virtual, para darle aún más realismo a su manejo, obteniéndose así posiblemente el SPV más completo y versátil que hay actualmente. Además, también se ha contribuido en la mejora de representaciones fosfénicas, al integrar el sistema con métodos de aprendizaje profundo que ayuden a destacar objetos de interés.

1.3. Objetivos y alcance

El objetivo principal es desarrollar un simulador de visión protésica simulada (SPV) inmersivo mediante un sistema de realidad virtual (Oculus Rift DK2), de forma que se puedan explorar entornos realistas creados por un ordenador navegando dentro de la escena, integrando el movimiento real de la cabeza del usuario en la simulación y permitiendo un desplazamiento por el escenario mediante un mando o joystick. La arquitectura propuesta se compone de dos ordenadores conectados de forma remota, uno basado en Windows que sirve de interfaz con las gafas e incluye parte del simulador de visión protésica y otro basado en Ubuntu, donde se va a generar el entorno virtual en tiempo real y se van a procesar las imágenes sintéticas para representar el entorno con fosfenos. Se pretende desarrollar un simulador lo más fiel posible a la realidad, en un entorno artificial y utilizar diferentes representaciones fosfénicas a partir de las imágenes sintéticas RGB-D usando técnicas de visión por computador.

Se ha escogido el simulador de robótica Gazebo para poder representar un entorno 3D de tal forma que se pueda girar la cabeza en todas direcciones y moverse a través de él, manteniendo limitaciones físicas para que existan colisiones con objetos. De esta manera se pretende que se pueda explorar el entorno de la forma más completa y realista posible, pudiendo visualizar todos los elementos desde diferentes perspectivas gracias al movimiento del sujeto.

Una visión general del método se muestra en la Figura 1.4. En primer lugar, el cliente de realidad virtual (VR) obtiene la orientación de la cabeza a partir de la información de una unidad de medida inercial (IMU) que está integrada en las Oculus, y se envía a través del *socket* al servidor Ubuntu. Esta orientación, junto con la traslación que comunica el usuario mediante el mando, es enviada por al simulador Gazebo a través de los canales de comunicación del sistema operativo robótico ROS (*topics*; ver Anexo III) que genera la imagen RGB-D correspondiente a esa localización en el mapa. Finalmente, el servidor transforma la imagen que capta la cámara virtual a una imagen de fosfenos y se manda la información de la luminosidad de cada uno de ellos al cliente de RV a través del socket para que pueda representar la imagen de fosfenos en el visor de realidad virtual.

Nótese que no hace falta enviar la imagen completa a través del socket. Al precalcular la posición de los fosfenos y su tamaño una única vez al iniciar el programa se puede transmitir únicamente el nivel de iluminación de cada fosfeno, aligerando el funcionamiento de éste para que la ejecución en tiempo real sea fluida.

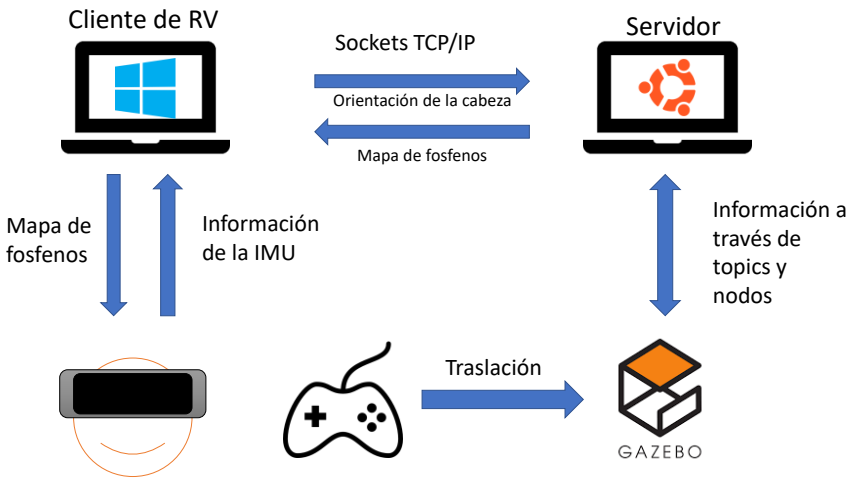
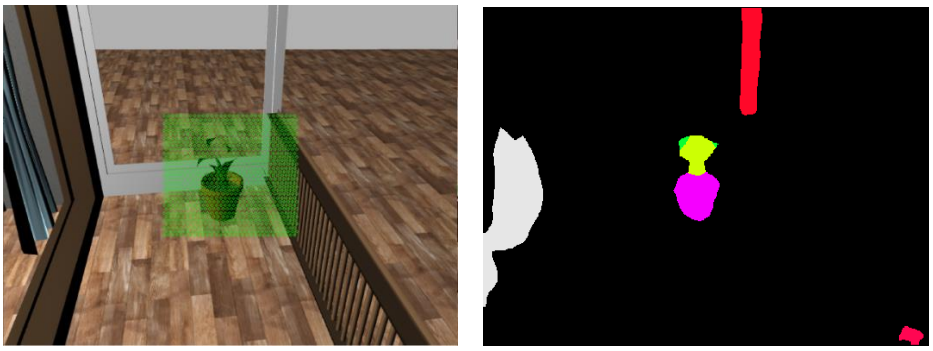


Figura 1.4. Esquema general del sistema

La novedad de este proyecto es que, a pesar de existir otros simuladores de prótesis, no existía ninguno que usase gafas de realidad virtual permitiendo una experiencia inmersiva en un entorno virtual 3D con posibilidad de exploración completa en traslación y rotación. Este trabajo ha sido creado con la idea de poder ser usado en un futuro para comparar diferentes modelos de representación de fosfenos y analizar el impacto de la traslación en el reconocimiento de objetos. El uso de un entorno simulado permite la realización de experimentos de forma sistemática y repetitiva de forma que se puedan obtener resultados con sentido estadístico.

Además, se ha introducido una nueva representación que permite destacar algunos objetos de interés, utilizando métodos modernos de deep learning de segmentación semántica. En la Figura 1.5. aparece la imagen de una planta y la segmentación obtenida a partir de ella. En la Figura 1.6.a. aparece el mapa de fosfenos obtenido mediante el método downsampling y en la Figura 1.6.b. el modo de visualización introducido resaltando el objeto:



a) Imagen original **b) Imagen segmentada**

Figura 1.5. Ejemplo de segmentación

a) Downsampling **b) Downsampling y segmentación**

Figura 1.6. Comparación entre métodos de representación de mapas de fosfenos con 20º de campo de visión

Cabe destacar que solo se visualiza la zona marcada en verde en la Figura 1.5.a. (en este caso corresponde a 20º de campo de visión) y que es más evidente que hay un objeto en la Figura 1.6.b. que en la Figura 1.6.a. Debido a la pobre resolución de las prótesis y pequeño campo de visión, a veces no resulta inmediato darse cuenta de lo que se tiene delante y esta representación ayuda a solventar ese problema.

1.4. Entorno de trabajo

Para conseguir una experiencia más inmersiva se han usado las gafas de realidad virtual Oculus Rift DK2 en este proyecto. Como se explica en [23], estas gafas pertenecen a Oculus VR y se trata del modelo Rift Development Kit 2 [24]. Fueron lanzadas al mercado en 2014 como segunda versión de un casco de realidad virtual pensado para la venta a desarrolladores. Tienen soporte técnico y un kit de desarrollo que facilita su programación, sin embargo, ya no se encuentran a la venta. Al tener disponible el sistema de software de Oculus se ha decidido trabajar con este modelo.

El kit incluye las propias gafas y una cámara externa para medir la traslación que en este proyecto no será necesaria dado que el simulador sólo contempla rotaciones con el usuario situado en el centro del entorno virtual.

Las gafas están compuestas, por la pantalla del visor por la que se muestra el entorno virtual, dos lentes asféricas, una para cada ojo, el hardware de procesamiento, una unidad de medición inercial (inertial measurement unit, IMU) y la carcasa que sujeta todo el conjunto. Una vista generalizada de su constitución puede observarse en la Figura 1.7.

Figura 1.7. Vista expandida de las gafas Oculus Rift DK2 [25]

La pantalla proviene de Samsung, y de hecho emplea la misma pantalla que monta el modelo Galaxy Note 3 [26]. Es una pantalla de 5,7” con una resolución de 1920 píxeles horizontales por 1080 verticales [27]. Como la imagen mostrada por pantalla está duplicada ya que muestra la imagen de salida para cado ojo, la imagen de salida debe ser 1080x960 píxeles.

Dado que la pantalla se encuentra muy cerca del ojo, se necesita utilizar un sistema de lentes que permita un enfoque correcto a la vez que muestre un campo visual elevado. Las lentes usadas son las de tipo asférico para evitar errores ópticos y reducir el número de lentes necesarias. El uso de estas lentes más una distorsión de tipo barrilete (Figura 1.8b) introducida por el "Software Development Kit" (SDK) permiten corregir las anomalías, y finalmente, junto al globo ocular, se corrige la distorsión introducida (Figura 1.9.).

(a) Imagen sin distorsión **(b) imagen con distorsión enviada a las Oculus**

Figura 1.8. Distorsión de tipo barrilete [2]

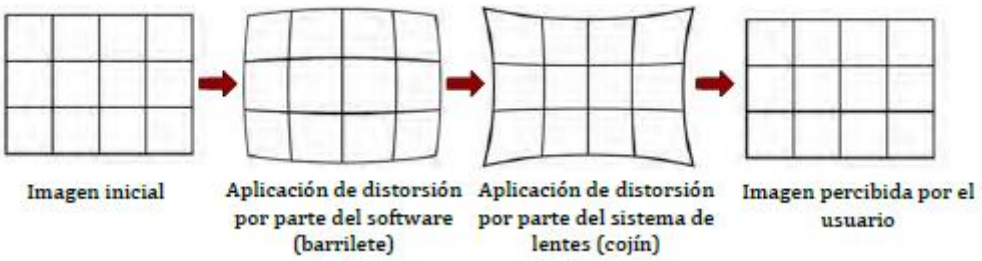


Figura 1.9.Corrección de la distorsión de tipo barrilete [23]

La implementación del interfaz de realidad virtual en el sistema Windows se ha programado en lenguaje C++ sobre el entorno de desarrollo Visual Studio 2013 utilizando la API del SDK de Oculus junto con las librerías OpenCV para facilitar el manejo de

imágenes. Desde su aparición en el año 2000 se han convertido en el estándar para las aplicaciones de visión por computador [28].

En cambio, el desarrollo en el servidor Ubuntu que se utiliza para la simulación de entornos virtuales, se ha realizado en C++ y Python dentro del entorno de Robot Operating System (ROS), un entorno para el desarrollo de software de robots [29], que da soporte para comunicar procesos (*nodos*) mediante canales denominados *topics*. (Para más detalle consultar Anexo III). Se han usado paquetes ya creados y herramientas como:

- RGBD-VIA: un simulador de visión protésica desarrollado en la Universidad de Zaragoza, que permite generar imágenes de fosfenos con una cámara RGB-D [14], [30].
- Gazebo: es un simulador 3D, cinemático, dinámico y multi-robot que permite realizar simulaciones de robots articulados en entornos complejos, interiores o exteriores [31]. Se suele usar en conjunto con ROS ya que ambos son desarrollados y distribuidos por Open Robotics.
- RVIZ: Herramienta de visualización de diferentes tipos de datos como imágenes, nubes de puntos, escáneres láser, mapas o modelos robóticos [14]. Su función es mostrar los mensajes con los que ROS trabaja, algunos de ellos pudiendo ser visualizados en 3D.
- DilatedNet [32], [33]: es una red neuronal pre-entrenada para segmentación semántica. Se quiere integrar en un *nodo* ROS para resaltar objetos de la escena.

Capítulo 2

Mapa de Fosfenos

Las prótesis visuales son la única manera que hay actualmente de ayudar a mejorar la visión en pacientes con enfermedades degenerativas ya que actualmente no existe tratamiento médico.

Los implantes retinales actuales constan de un medio que captura la información visual y de un procesador la convierte en patrones de estimulación que se transmiten al implante. La imagen resultante de este proceso corresponde a un mapa de fosfenos de muy baja resolución espacial y con un rango dinámico bajo [34].

2.1. Tipos de Implantes

Como se explica en [13], existen diferentes tipos de prótesis según el problema que cause ceguera.

2.1.1. Implantes retinales

Estimulan las células de la retina. El conjunto de electrodos puede ser epiretinales, subretinianos o supracoroideos.

Epiretinales: dentro de este tipo se encuentra la prótesis más usada mundialmente, el Argus II perteneciente a la empresa estadounidense Second Sight Medical Products, Inc, el cual, como se puede observar en la Figura 2.1. consta de una cámara colocada en el centro de unas de gafas, una bobina y una unidad de procesamiento de vídeo portátil que transmite inalámbricamente impulsos electrónicos al implante de 60 electrodos [35]. Es uno de los dos modelos de prótesis aprobados por la Unión Europea para el tratamiento de retinitis pigmentosa [36]. Tiene un ángulo de visión máximo de 20º

en diagonal. La separación entre los centros de los electrodos es de 525µm, que corresponde a unos 1,88º de campo visual, suponiendo que 1º equivale a 280 µm de longitud en la retina [37].

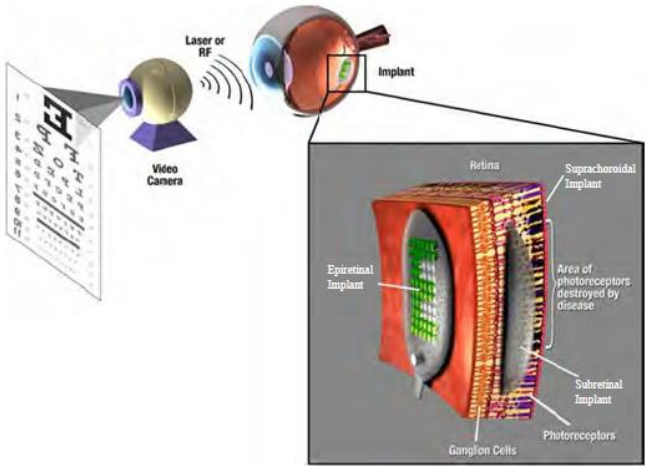


Figura 2.1. Implante Argus II [38]

Subretinales: Aquí se encuentra la otra prótesis más usada junto a la anterior, el Alpha-IMS de Retina Implant AG. Se trata de un implante que no posee una cámara como en el anterior caso si no que dispone de 1500 fotodiodos que capturan la luz proyectada sobre la retina y la transforman en señales eléctricas. El implante tiene un tamaño de 3 x 3 mm y los electrodos tienen un diámetro de 50µm, con una distancia entre centros de electrodos de 70 [39]. En la Figura 2.2. se compara el campo visual de ambos implantes.

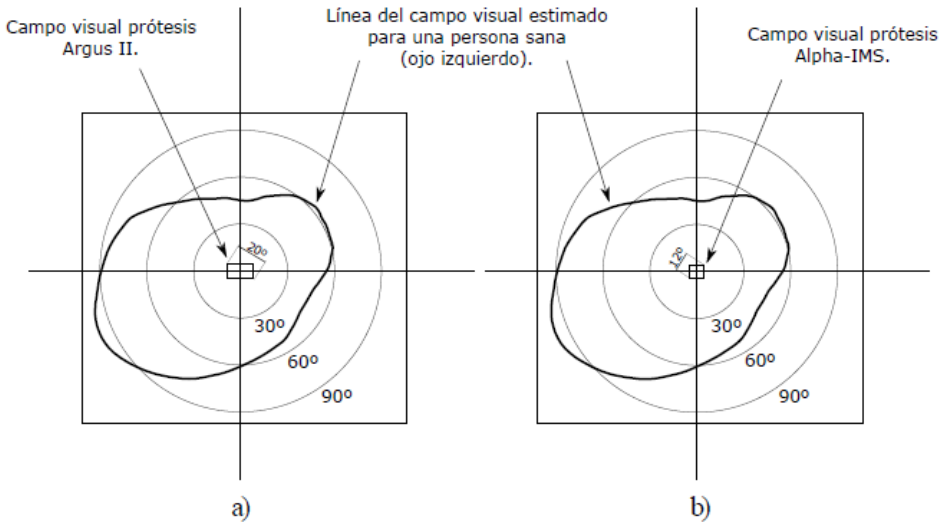


Figura 2.2. Comparación entre los dos modelos de prótesis [13]

La resolución del Alpha es mayor que la del Argus pero, sin embargo, algunas investigaciones señalan que no implica una mejora significativa [40]. La razón puede ser que el Argus permite ajustar para cada electrodo las variables mientras que en el Alpha este control es más limitado [41].

Dadas estas limitaciones, es deseable buscar maneras más simples de representar el entorno gracias a un procesamiento inteligente de la imagen para dar más información con pocos fosfenos. Esto no se puede hacer con el Alpha-IMS puesto que no tiene cámara ni procesador, pero sí que se podría con la Argus II.

2.1.2. Nervio óptico

Este tipo de implante tiene menos criterios de exclusión que el implante de retina, por lo que pueden llegar a usarlo más pacientes[42]. La primera estimulación eléctrica del nervio óptico fue conseguida por [43]. Se demostró que los fosfenos podían reproducirse con estimulaciones seguras, y predecir la ubicación y el tamaño de las percepciones en función de los parámetros de estímulo [44]. Con entrenamiento, los receptores pueden reconocer y orientar formas complejas [45] y realizar la localización y discriminación de objetos [46].

2.1.3. Córtex Visual

Las primeras estimulaciones eléctricas de fosfenos fueron en esta área. En 1968, [4] utilizó una serie de receptores de radio conectados a electrodos en contacto con el polo occipital derecho de un paciente de 52 años, se descubrió que al estimular varios electrodos simultáneamente, el paciente podía ver patrones simples, y al mover los ojos, los fosfenos se movían acorde.

Generalmente los fosfenos desaparecían inmediatamente, pero en algún caso lograban permanecer hasta 2 minutos después de un estímulo fuerte.

2.2. Representación de Fosfenos

A continuación, se explica, de manera general, como funcionan las funciones de fosfenos utilizadas en RGBD-VIA [30], y actualizadas en el cliente de realidad virtual para que la representación de los mismos replique perfectamente el mapa de fosfenos.

A la representación de cada fosfeno se le llama *Sprite*, es como un pequeño circulo luminoso que aparece en el campo de visión. Algunos estudios han demostrado que no

siempre es redondo y que puede tener variaciones de forma o incluso color [47], sin embargo para este trabajo se han ignorado esas anomalías dado que no son muy comunes y se ha decidido representarlos mediante círculos cuya iluminación siga una distribución gaussiana de dos dimensiones de manera que en el centro se obtiene más intensidad que la parte exterior de acuerdo con estudios de otros SPV [10]. El tamaño del fosfeno típicamente se mide en ángulos de campo de vista, y para nuestro trabajo hemos elegido un tamaño de 0.6º. Los fosfenos solo tendrán 8 niveles de luminosidad posibles [10], desde completamente apagados hasta completamente iluminados como muestra la Figura 2.3:

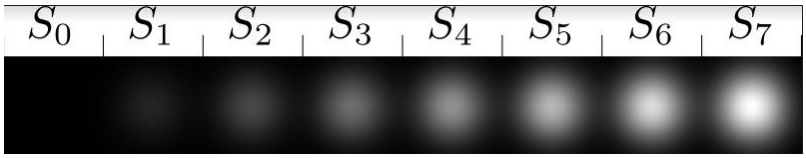


Figura 2.3. Visualización de los diferentes niveles de luminosidad de los *sprites*

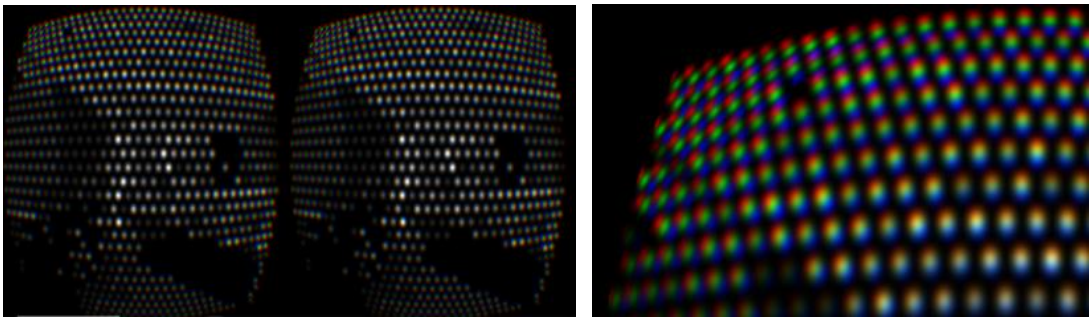
Para calcular la distribución de fosfenos en la imagen, se parte del tamaño de la imagen del visor, que es el mismo tamaño que la imagen que muestra por pantalla el SPV de RGBD-VIA, y del número de fosfenos que se quiera representar (se visualizan 1.000). Se calcula la separación entre fosfenos (delta) para que quepan el número de fosfenos consignados en el tamaño de imagen que queremos. Se va a intentar llenar la imagen del visor distribuyéndolos uniformemente según el trazado que se desee. Se ha escogido disposición hexagonal ya que ciertos estudios afirman que en los implantes retinales la localización de los fosfenos es bastante regular y además este tipo de distribución permite una mayor densidad de fosfenos [48], pero se podría escoger la disposición rectangular.

Una vez que se conoce la delta de los fosfenos, se calcula su posición a partir del tamaño de la imagen y el número de fosfenos. Adicionalmente, se devuelven en una matriz los rayos proyectantes asociados a cada fosfeno que pueden calcularse ya que se conoce el tamaño de imagen, el campo de vista que abarca, y la posición de cada uno de ellos.

Para dar más realismo, también se puede añadir ruido gaussiano a las posiciones de los fosfenos o aplicar dropout, es decir que algunos fosfenos no se iluminen, ya que son eventos que se ha comprobado que suceden en implantes reales [49].

Estas funciones se ejecutan una única vez al inicio del programa ya que el tamaño y posición de los fosfenos, así como su forma icónica no varía a lo largo de la ejecución del programa, solo se actualiza la luminosidad de cada uno de ellos.

En el caso del cliente de realidad virtual, al usar las gafas Oculus se produce una aberración del color blanco en la representación monocromática con lente asférica. Los mapas de fosfenos no presentan color, son monocromáticos, pero al introducir la distorsión de tipo barrilete explicada en el apartado 1.4. y mirar el visor de las gafas, se puede apreciar que aparecen trazas de colores en fosfenos que no deberían estar, sobre todo en la periferia (Figura 2.4.b). Por eso se ha optado por representar los fosfenos en color verde (Figura 2.5.) por ser el color primario de más resolución.



(a) Representación con 1.000 fosfenos **(b) Ampliación de la zona donde los colores se ven con mayor claridad**

Figura 2.4. Aparición de colores por la distorsión [2]

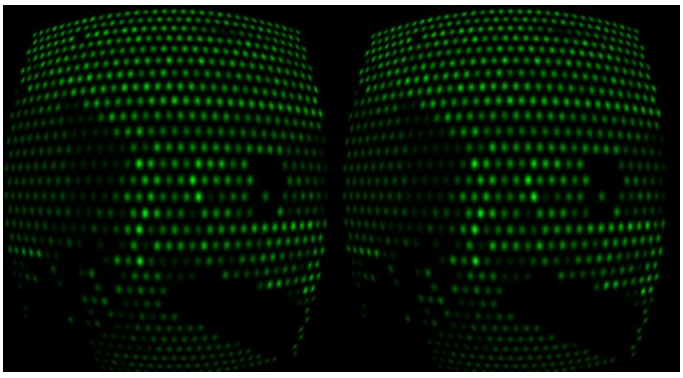


Figura 2.5. Representación de fosfenos en color verde en el visor de RV [2]

Por último, la visualización final de los fosfenos consiste en calcular para cada fosfeno, cuál de los 8 niveles de luminosidad posee. Esta información se va a almacenar en el vector *phospheneFlag*, de tamaño igual al número de fosfenos representados, cuyos valores son enteros dentro del rango [0,7]. Las distintas representaciones de mapas de fosfenos dan valores a ese vector.

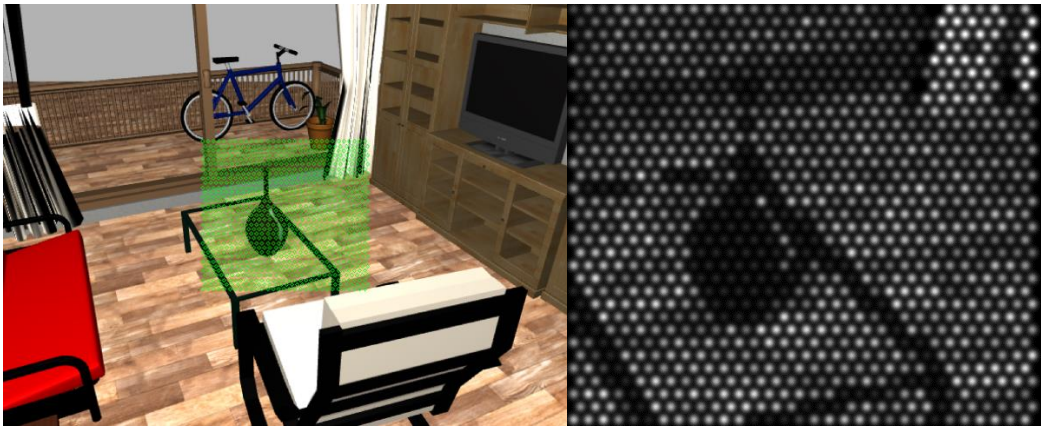
2.2.1. Representaciones simuladas

Al trabajar con fosfenos, el valor de cada uno de ellos se puede calcular de diferentes maneras. RGBD-VIA permite varios modos de visualización distintos que se explicarán a continuación.

Downsampling

Este método consiste en reducir la resolución de color de la imagen a 8 niveles de gris y reducir su resolución espacial.

La imagen obtenida de Gazebo ha sido inicialmente convertida a escala de gris con 8 niveles de grises, que es un rango dinámico aceptable para una prótesis actual. En la Figura 2.6. se presenta un ejemplo de esta representación. Se puede observar que cambia el campo de vista con respecto al de la cámara, que será generalmente mayor dadas las limitaciones de las prótesis. No obstante, al conocer la calibración de la cámara, se pueden reproyectar los rayos proyectantes de cada fosfeno que se han calculado antes en la imagen (y son fijos) y con eso sacar el valor RGB que se utiliza para calcular el nivel de luminosidad.



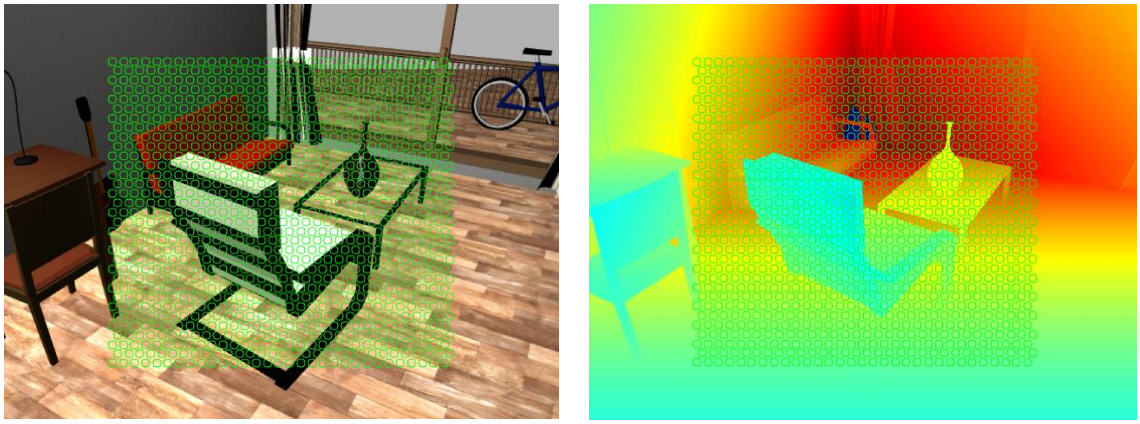
a) Imagen de la cámara de Gazebo con 20º de campo de visión **b) Representación en fosfenos**

Figura 2.6. Representación Downsampling

Modo de profundidad

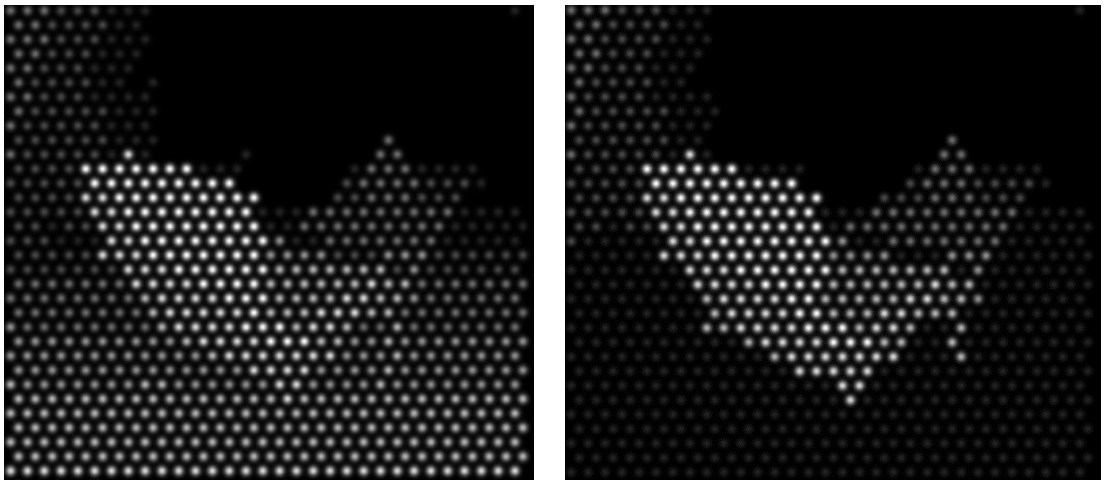
Al tener en Gazebo una cámara RGB-D, se puede utilizar el mapa de profundidad y usarlo para calcular el valor de la intensidad de los fosfenos. Los objetos que se encuentran más cercanos a la cámara aparecerán con más luminosidad que aquellos que se encuentran más lejos. Además, existe un umbral que se puede ajustar a partir del cual ya no se visualizan los objetos que se encuentren más alejados. Adicionalmente, cambia el

escalado, es decir, se tienen 8 niveles de luminosidad y si el umbral define que se visualiza hasta 2m, cambiará de nivel cada 2/8 =0.25m. La imagen de profundidad se puede ver en la Figura 2.7.b. y su mapa de fosfenos en la Figura 2.8.a:



a) Imagen a color **b) Imagen de profundidad**

Figura 2.7. Imagen de referencia de la cámara Gazebo para el modo de profundidad



a) Mapa de fosfenos del modo de profundidad **b) Mapa de fosfenos del modo de profundidad aumentada**

Figura 2.8. Representación del modo de profundidad para un campo de visión de 40º

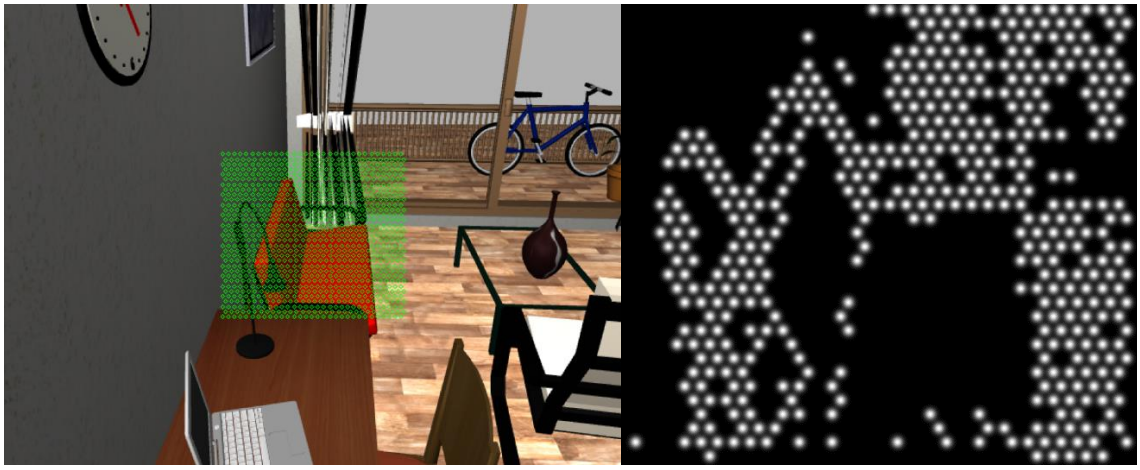
Se ha aumentado el campo de visión a 40º para que se aprecie mejor. Se puede observar que la mesa se visualiza con menos intensidad que el sillón porque se pierde luminosidad cuanto más lejos estén los objetos. Esta representación puede ser de utilidad para detectar presencia de objetos y evitar colisiones.

También se tiene otro modo de visualización que es igual que el anterior, pero extrayendo el suelo [50], esto es posible porque es una simulación y se conoce siempre la posición del robot con respecto al mundo, pero en un entorno real no es tan sencillo de detectar, debería estar calculándose continuamente. El trabajar con simulación se puede

considerar este cálculo perfecto. En la Figura 2.8.b. se puede apreciar que el suelo ya no se visualiza con respecto a la Figura 2.8.a, lo que puede ayudar además a distinguir objetos en la escena (como el sillón).

Método de Canny

Se utilizan métodos de procesamiento de imagen para extraer los bordes (puede ser en la imagen a color o en la de profundidad) y resaltar dichos bordes mediante fosfenos luminosos. En la Figura 2.9. se puede observar el resultado de dicho procesamiento:



a) Imagen de la cámara de Gazebo **b) Representación en fosfenos**

Figura 2.9. Representación del método de Canny

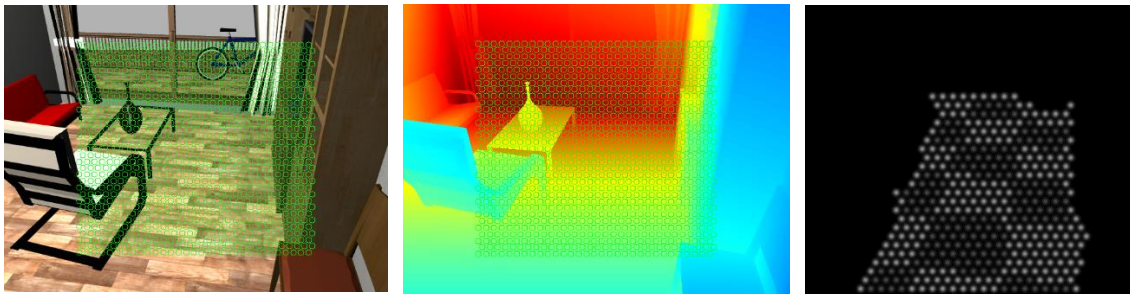
Método del suelo de ajedrez

Al tener con las prótesis un campo de vista tan limitado, es difícil transmitir la sensación de profundidad y avance en una escena. Asumiendo que no se tiene rango suficiente como para mostrar la profundidad variando niveles de intensidad, una representación mínima es pintar "baldosas" de distintos niveles de intensidad, como un tablero de ajedrez, que ayuda a dar esa sensación de profundidad [30]. También ayuda a orientarse según las direcciones principales que definen las líneas del suelo.

El patrón de ajedrez sólo se muestra sobre el espacio libre del suelo sobre el que puede caminar el sujeto, por lo que en cuanto la cámara de profundidad detecta que hay un obstáculo delante, el suelo deja de dibujarse. Así, sirve, además, para permitir navegar sin chocar con nada.

Este método además mantiene cada baldosa fija en su posición, con lo que al mover el robot por el entorno se puede tener una idea de cuanto se ha avanzado. Para ello

hay que conocer cuánto se ha movido en la escena. En este caso, gracias a que se simula en Gazebo, se puede saber automáticamente. En un sistema real, habría que añadir, adicionalmente, algún tipo de SLAM o sistema de odometría visual que ayudará a saber cuánto se ha desplazado. En la Figura 2.10. se puede ver que el sistema de representación omite el sofá y muestra el espacio libre del suelo con el patrón de ajedrez.

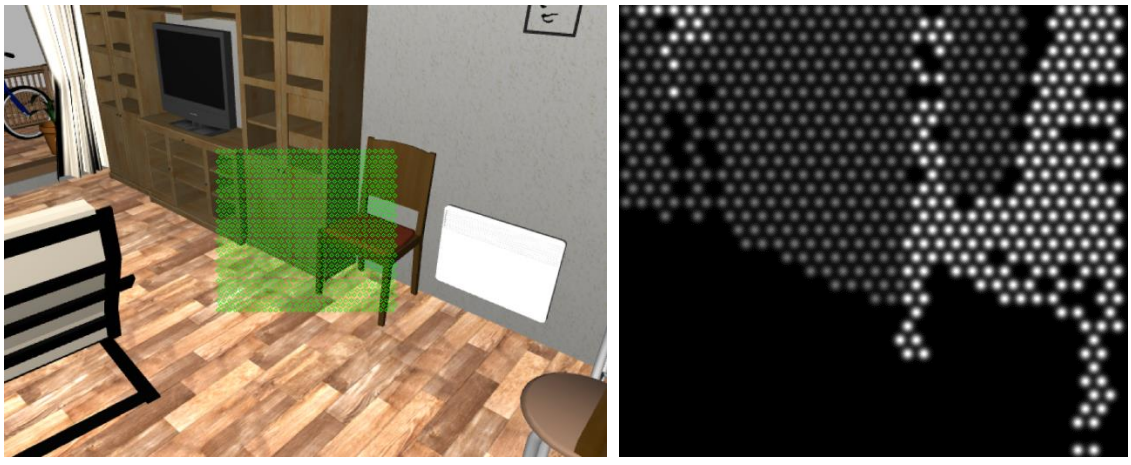


a) Imagen de la cámara de Gazebo **b) Imagen de profundidad** **c) Representación en fosfenos**

Figura 2.10. Representación del suelo de ajedrez

Método de detección de obstáculos y paredes

Se utiliza la nube de puntos para eliminar el plano horizontal del suelo, pintar los planos verticales en gris y los planos horizontales que se encuentran por encima del suelo y el resto en blanco (Figura 2.11), por lo que se pueden llegar a visualizar objetos, sin embargo, este criterio es fundamentalmente geométrico y no detecta e identifica los objetos como tal. En el Capítulo 5 se presenta un nuevo modo de visualización basado en criterios de segmentación semántica.

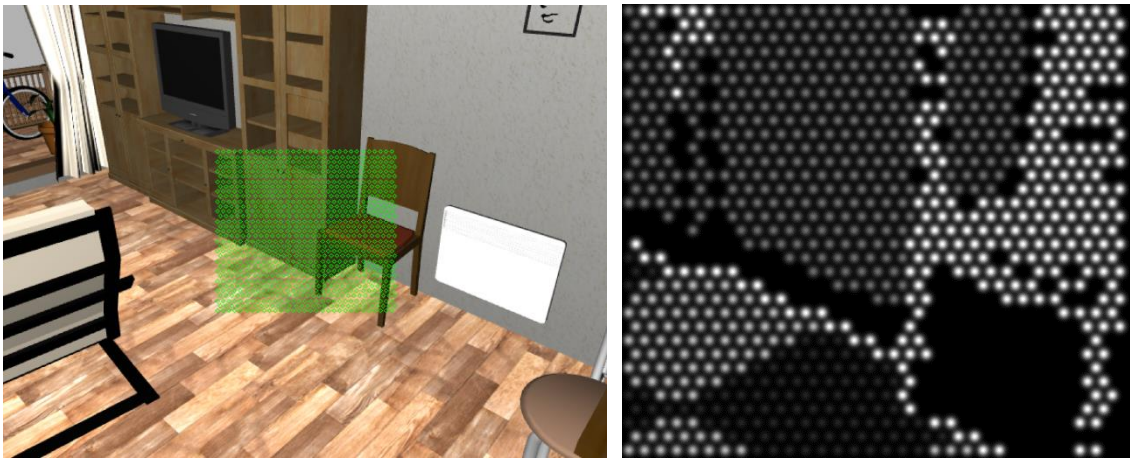


a) Imagen de la cámara de Gazebo **b) Representación en fosfenos**

Figura 2.11. Representación del método de detección de obstáculos y paredes

Método combinado de suelo de ajedrez y detector de paredes y obstáculos

Se pueden combinar los métodos anteriores y se obtiene el resultado de la Figura 2.12:



a) Imagen de la cámara de Gazebo **b) Representación en fosfenos**

Figura 2.12. Representación del método combinado de suelo de ajedrez y detector de paredes y obstáculos

Capítulo 3

Implementación del mundo virtual

En este capítulo se van a abordar los diferentes aspectos introducidos para mejorar el mundo virtual, pasando por una introducción en la que se explica el modelo de persona utilizado para navegar a través del entorno, la situación de partida de dicho modelo, las modificaciones introducidas para lograr un movimiento más completo de cabeza y los nuevos entornos más realistas.

3.1. Modelo del sujeto

El robot usado para modelar a la persona que se mueve por el entorno es el robot *blindbot*, consiste en una readaptación de Turtlebot [51], al cual se le hizo diferentes ajustes para que parezca una persona. El robot Turtlebot original consta de una estructura montada sobre una base de Roomba y que integra sensores de odometría y la cámara Asus Xtion Pro Live, a este robot se le añadió un modelo de persona y se colocó la cámara a la altura de los ojos como se puede ver en la Figura 3.1.a:

a) **Modelo blindbot** **b) Representación de los diferentes sistemas de coordenadas**

Figura 3.1. Modelo blindbot y los sistemas de referncia

Como se puede observar en la Figura 3.1.b, se han representado los sistemas de coordenadas del mundo, la base del robot y la cámara. Cada elemento que es un cuerpo rígido aparece como un *link*, y las *joint* unen los distintos *links* y describen la cinemática y dinámica de la articulación. Para más información consultar el Anexo III.

En la Figura 3.2. se muestra el esquema de árbol de transformaciones entre los distintos links.

Este esquema indica la relación desde el origen de coordenadas del mundo simulado (*world*), a la base del robot, cuya posición y orientación relativa entre ambas se pueden conocer al estar en un entorno de simulación. Acoplados a la base están los diferentes sensores y elementos móviles (parachoques, ruedas, etc.) hasta llegar a la cabeza del modelo de persona introducido (*head_link*), donde se coloca la cámara (*camera_link*), la cual tiene información de color y de profundidad. Aunque en el esquema no aparece especificado, la joint que define la articulación entre *head_link* y *camera_link*, introduce el movimiento de cabeceo del robot *blindbot*, pero no tiene los dos giros restantes para poder modelar una rótula esférica, como es el movimiento real que realizan las personas.

Este movimiento de cabeza que tiene y el desplazamiento de la base del robot se pueden controlar mediante mando o teclado, gracias a otros *nodos* que permiten publicar en los topics de movimiento. A su vez, la cámara publica en diferentes topics las imágenes de la cámara RGB y de profundidad, así como nubes de puntos, que es lo que luego se usa en RGBD-VIA como entrada para realizar todo el procesamiento de generación de fosfenos.

Es importante resaltar, que la cámara del robot usa el mismo tipo de mensajes y topics que usaría una cámara real, por lo que sería prácticamente directo usar todo el trabajo realizado con una cámara en vez de en un mundo virtual.

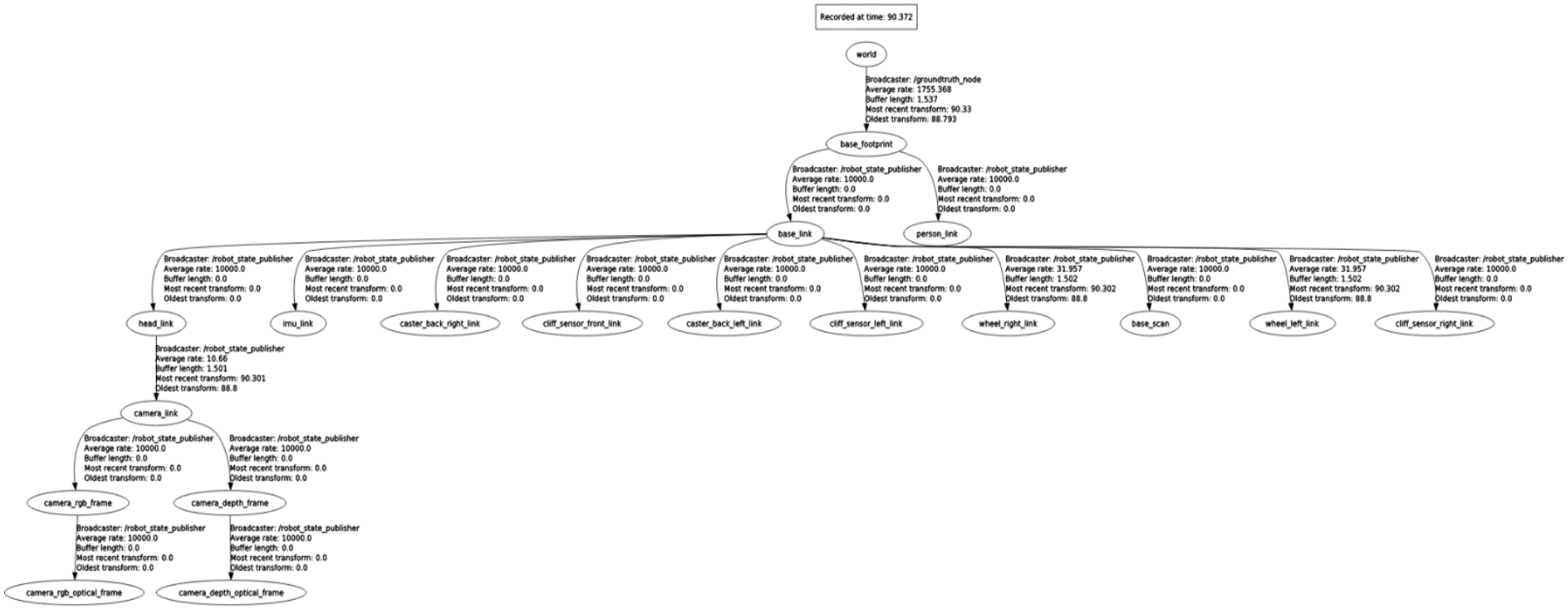


Figura 3.2. Esquema de árbol de transformaciones entre los distintos elementos

3.2. Implementación de los ángulos de giro

La versión original del simulador estaba pensada para un uso con monitor y mando de forma que solo permitía girar la cabeza de arriba a abajo (cabeceo). Para que la experiencia con el nuevo sistema de realidad virtual sea inmersiva y plenamente funcional es necesario modelar los tres grados de libertad de la orientación de la cabeza. Esta extensión del modelo de sujeto permite que se pueda explorar la escena al completo girando la cabeza.

Para conseguir implementar los 3 giros de las gafas, se ha modificado el fichero URDF (Anexo III) que se utiliza en Gazebo, y se crean tres links, con sus respectivas joints, una para cada ángulo, que además deben tener cierta masa e inercia para que funcionen de forma correcta, ya que Gazebo ignora aquellos links que no tienen propiedades inerciales, y se concatenan entre sí. Los cambios introducidos se pueden ver de forma esquemática en la Figura 3.3:

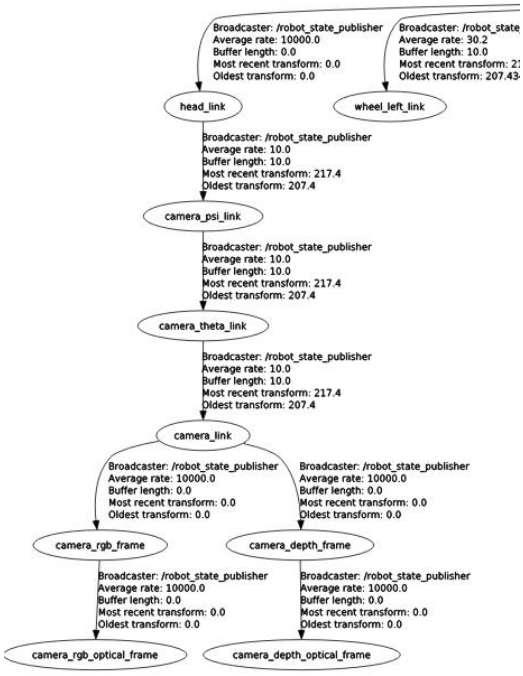


Figura 3.3. Esquema de árbol con modificaciones para los nuevos giros

Se observó que había cierto retardo que hacía que el movimiento fuese poco natural, ya que Gazebo es un simulador físico y el movimiento se hacía con un controlador que funciona de manera realista. Este es un retardo bastante apreciable lo que no es

deseable ya que los movimientos de cabeza de las personas deben ir acorde con lo que se visualiza en cada momento.

Las joints deben de ser de tipo *continuous* para que Gazebo no las mantenga fijas como pasaría con las de tipo *fixed*. Además, se necesita que esas joints se muevan de forma inmediata a la posición deseada, sin ningún controlador de bucle cerrado, publicando los ángulos en los topics de forma que el movimiento sea instantáneo para que cuando la persona que se ponga las gafas mueva la cabeza no note ningún tipo de retardo en la visualización de la imagen del visor. Por este motivo, se ha implementado un nuevo plugin [52] que fuerza a todas las joints a mantenerse en el último valor que se ha publicado en el topic. De esta forma se consigue que la respuesta a los movimientos sea inmediata. Este plugin se suscribe a cada joint de los ángulos *roll-pitch-yaw*, generando un topic de tipo Float32, donde el valor deberá ser introducido en radianes. Como se detalla en el apartado 4.2, el nodo *server.cpp* recibe por el socket los ángulos que describen la orientación de las gafas y escribe en esos topics directamente el ángulo al que se tienen que mover. En la Figura 3.4. se indican los ángulos de las gafas:

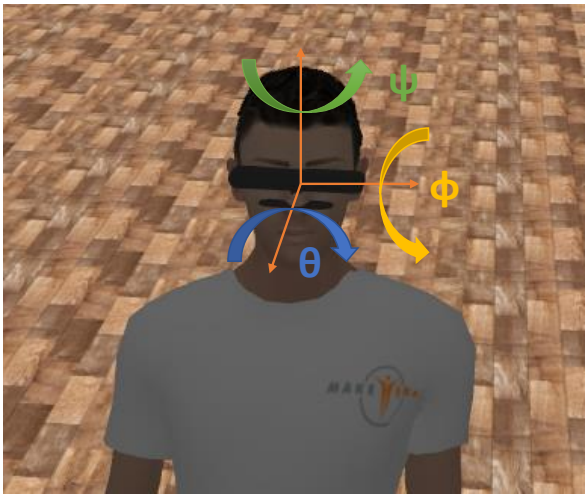


Figura 3.4. Ángulos roll-pitch-yaw

3.3. Implementación de nuevos mundos virtuales

Para que la experiencia con el simulador sea más realista, se han añadido entornos de Sweethome3D, una aplicación libre de diseño de interiores que ayuda a generar escenarios sobre un plano 2D, con posibilidad de vista previa en 3D [53]. Este programa permite exportar estas casas más realistas a formato .obj y con el programa Blender se puede convertir a un archivo tipo collada .dae, un tipo de archivo que Gazebo es capaz de

manejar. Además, conserva las texturas y colisiones y este programa dispone de una galería con bastantes casas prediseñadas, ofreciendo más posibilidades de diseño que el editor de Gazebo, además de más variedad de objetos realistas y detallados tal y como se puede ver en la Figura 3.5.

Figura 3.5. Ejemplo de casa en Sweethome3D

Figura 3.6. Casa exportada en Gazebo

El escenario de la Figura 3.5. ha sido exportado a Gazebo (Figura 3.6.) y se le ha añadido un suelo, iluminación y modelos de personas para crear un escenario realista. En las Figuras 3.7. y 3.8. se pueden ver más ejemplos:

Figura 3.7. Ejemplo de otra casa de Sweethome3D

Figura 3.8. Casa de la Figura3.7. en Gazebo

Capítulo 4

Intercomunicación entre equipos

En este proyecto se ha escogido comunicar ambos equipos (servidor y cliente) a través de *sockets* con protocolo de internet TCP/IP, ya que es una manera rápida de intercambiar cualquier flujo de datos [54]. Para ello, es necesario que un equipo sea capaz de localizar a otro (gracias a su IP y puerto).

A continuación, se explica detalladamente cómo se realiza dicha conexión y a partir de la estructura de socket descrita en el Anexo II, cómo se gestiona el envío y recibo de información.

4.1. Cliente de realidad virtual con visión protésica

En el ordenador que sirve de interfaz con el sistema de realidad virtual, se ha implementado la parte del socket que realiza la labor de cliente, indicando la IP y el puerto al que se desea conectar, que pertenecerá al equipo Ubuntu, para poder enviar la información de la orientación de las gafas y recibir la trama de intensidades del mapa de fosfenos del servidor donde se ejecuta el entorno virtual. Además, la comunicación entre equipos, este módulo configura la malla de fosfenos en el visor de las gafas, para que solo sea necesario recibir en tiempo real la información de luminosidad de cada uno de ellos desde el servidor para visualizar correctamente el mapa de fosfenos.

Para ello, se inicializan variables iniciales, como el número de fosfenos, (en este caso 1.000), se inicializan las funciones de posición y tamaño de los fosfenos (gracias a este cálculo inicial y único es capaz de funcionar a tiempo real ya que se calcula una única vez) y se inicializa el socket tal y como se ha explicado en el Anexo II, seleccionando un

puerto libre y la IP del servidor, que siempre es fija ya que se trata de un ordenador de un laboratorio de la universidad.

Tras inicializar las funciones, variables y el socket, en tiempo real la IMU de las gafas recoge la información del movimiento de cabeza realizado por el usuario y almacena la orientación de la cabeza codificada en forma de cuaternios. A través de la ecuación 1 se obtiene la matriz de rotación.

$$R = \begin{pmatrix} 1-2*(q_y*q_y+q_z*q_z) & 2*(q_x*q_y-q_z*q_w) & 2*(q_x*q_z-q_y*q_w) \\ 2*(q_x*q_y-q_z*q_w) & 1-2*(q_x*q_x+q_z*q_z) & 2*(q_y*q_z-q_x*q_w) \\ 2*(q_x*q_z-q_y*q_w) & 2*(q_y*q_z-q_x*q_w) & 1-2*(q_x*q_x+q_y*q_y) \end{pmatrix} \quad (1)$$

La información de orientación se va a enviar al servidor en forma de ángulos *roll-pitch-yaw*, ya que esta descripción corresponde directamente los giros de cabeza que se implementan en el modelo mecánico de sujeto en Gazebo. Los ángulos de *roll-pitch-yaw* se calculan según las ecuaciones 3, 4 y 5, considerando cada término de la matriz de rotación según la ecuación 2

$$R = \begin{pmatrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{pmatrix} \quad (2)$$

$$\phi = atan2(n_y, n_x) \quad (3)$$

$$\theta = atan2(-n_z, n_x cos\phi + n_y sen\phi) \quad (4)$$

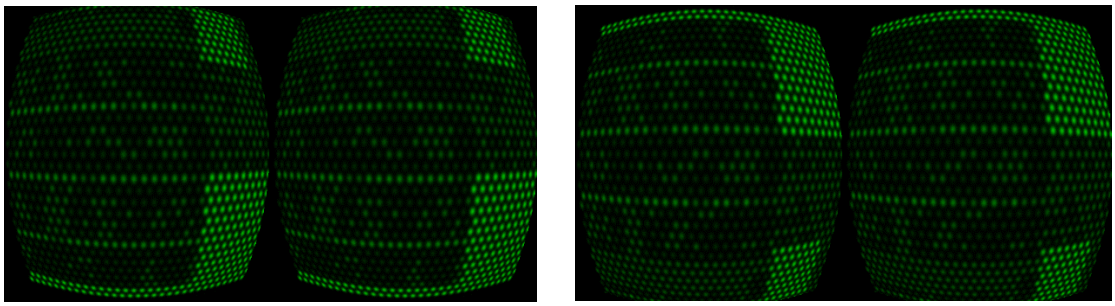
$$\psi = atan2(-a_y cos\phi + a_x sen\phi, o_y cos\phi - o_x sen\phi) \quad (5)$$

Para que los ángulos φ y ψ correspondan con los de los joints que modelan el movimiento de la cabeza en Gazebo (sección 3.2.) se les cambia el signo. Estos tres ángulos se envían a través del socket en una misma trama de datos.

Por otra parte, del servidor se recibe la información de la luminosidad de cada fosfeno del cual se ha precalculado su posición y tamaño. Esta información llega mediante un vector, de tamaño igual al número de fosfenos, en el que cada elemento indica el nivel de luminosidad entre 0 y el máximo (en nuestro caso entre 0 y 7) del fosfeno correspondiente. La imagen en el visor de las Oculus aparece del revés (Figura 4.2.a), y para solventarlo se reordena el vector de luminosidades (Figura 4.2.b.). Como se comentaba en el Capítulo 2, en las Oculus se ha decidido representar los fosfenos en verde porque debido a la distorsión introducida en las gafas para que al usarlas se perciba la

imagen sin distorsión como en la Figura 4.1, si se usase color blanco aparecerían aberraciones del color.

Figura 4.1. Visualización imagen en RGBD-VIA



a) Fosfenos ordenador tal cual llegan **b) fosfenos reordenados**

Figura 4.2. Ordenación de los fosfenos en el visor de RV

4.2. Interfaz Socket/ROS

Se ha creado un *package*, llamado *oculus*, para crear un *nodo* llamado server.cpp que sea a la vez *Publisher* y *Subscriber* y que además contiene la parte del socket que realiza la labor de servidor al que se conectará el cliente de realidad virtual. El esquema del funcionamiento de éste se describe en la Figura 4.3:

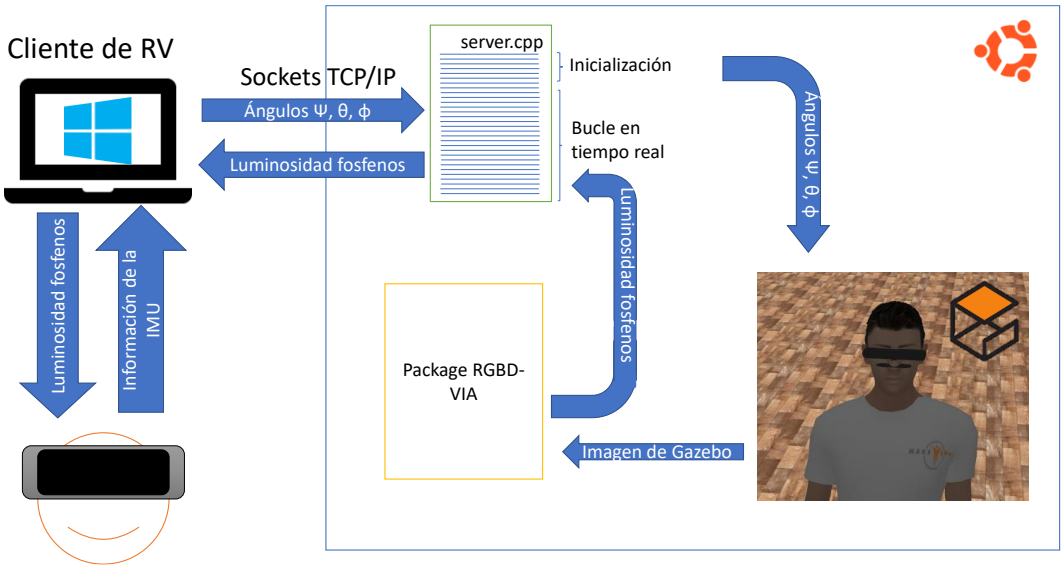


Figura 4.3. Esquema global de funcionamiento del SPV

Se inicializa el socket tal y como se ha explicado en el Anexo II, en la parte de la creación del servidor. También se crea un identificador para generar un *nodo* y se inicializan tres *Publisher* cada uno con el topic en el que se van a publicar los ángulos *roll-pitch-yaw* correspondientes que se reciban del socket. Por otra parte, el *Subscriber* leerá el topic que contiene el vector de la luminosidad de los fosfenos, creado dentro del *package* RGBD-VIA [30], ya que es el simulador de visión fosfénica que se usa en este equipo, y por lo tanto el que tiene todo el procesamiento de la creación y visualización de fosfenos, y precisamente se necesita recibir la información de la iluminación de cada fosfeno de dicho simulador.

La trama con los ángulos de orientación de la cabeza enviada por el cliente de realidad virtual es interpretada y publicada en tres topics ROS, uno para cada ángulo, que serán leídos por el modelo de sujeto en Gazebo. Este modelo es el que se utiliza para simular el movimiento de un individuo por el entorno virtual. También se llama a una función *Callback* que permite leer los topics a los que se suscribe este nodo, en este caso la información de la luminosidad de los fosfenos, y se guarda en un vector de enteros, ya que solo tienen 8 niveles de luminosidad que se manda por el socket hacia el equipo Windows.

En las Figura 4.4. se indica un esquema que ayuda a la comprensión del funcionamiento de este archivo.

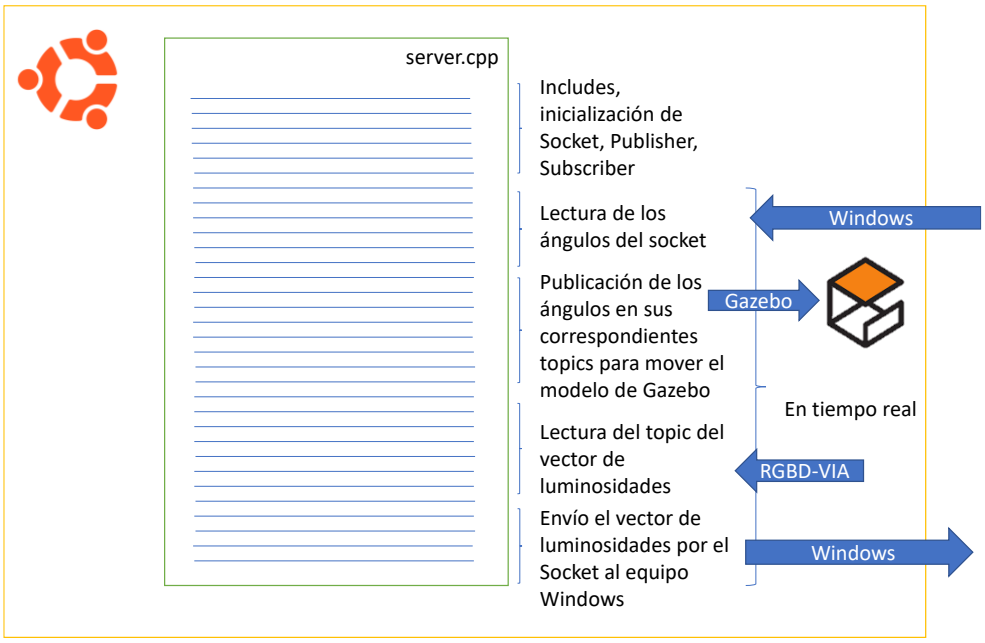


Figura 4.4. Esquema del nodo server.cpp

Capítulo 5

Segmentación semántica de objetos

Uno de los grandes problemas de las prótesis visuales es la baja resolución y bajo rango dinámico de la imagen, por eso es tan importante los SPV, para poder probar de forma eficaz diferentes representaciones, densidad de fosfenos, número, campo de visión, etc. para encontrar la mejor solución. El uso de un simulador de prótesis visual como este permite simular de forma realista la sensación percibida por un usuario de prótesis visual cuando se explora el entorno mediante rotaciones de cabeza y desplazamientos por el mismo.

Se ha implementado un *nodo* ROS que ejecuta una implementación en pytorch de modelos de segmentación semántica con el dataset de escenas MIT ADE20K [55],[56]. ADE20K es el conjunto de datos de código abierto más grande para la segmentación semántica y el análisis de escenas, lanzado por el equipo de MIT Computer Vision. Se usa la red neuronal pre-entrenada DilatedNet [32], [33] que es capaz de detectar de manera muy eficaz una gran variedad de objetos, 150 en total, aunque para el propósito de este SPV se ha reducido su número a detectar ya que solo se van a simular espacios interiores y varios de los objetos que detecta son de exteriores (como un tren, un lago, etc.) y podrían dar lugar a confusión. Por ello, para asegurar un mejor resultado se han descartado.

Se ha implementado un *nodo* ROS en python que integra la inferencia de esta red en tiempo real. Inicialmente se declaran las librerías necesarias como pytorch, numpy y el código para el modelo de segmentación y sus utilidades para visualizar los datos.

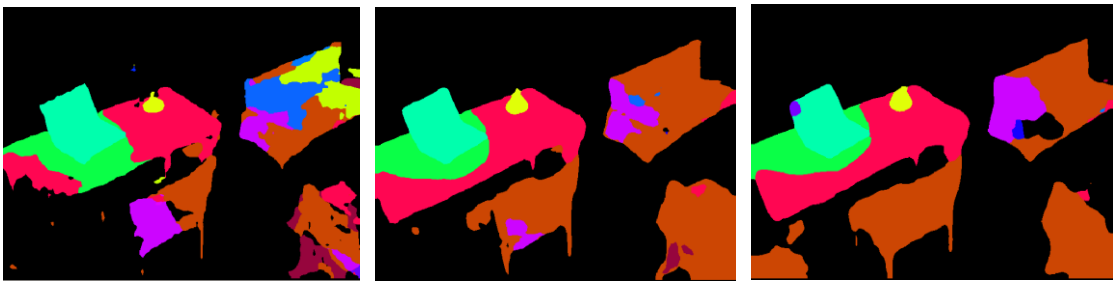
A continuación, se carga el modelo de segmentación pre-entrenado, se coloca en GPU y se define que solo se desea hacer inferencia, no entrenamiento. Además, se

suscribe al topic que se desea leer (el que publica la cámara del robot) y se generan los topics en los que se quiere publicar el resultado, en este caso se han definido dos, uno a color para visualizar exactamente el código de colores y otro en blanco y negro.

Una vez inicializado el modelo y los topics comienza el bucle principal del nodo que debe ser lo suficientemente rápido para ejecutarse en tiempo real.

Se lee el topic de la cámara de Gazebo y se convierte a tensor. Se pasa por el modelo de segmentación un total de tres veces, cambiando el tamaño de imagen para poder obtener una mejor segmentación final. En la Figura 5.2 se compara el resultado de ejecutar el modelo una vez a cinco veces con respecto a la imagen de referencia de la Figura 5.1.

Figura 5.1. Imagen de la cámara de Gazebo



a) Resultado con una pasada **b) Resultado con tres pasadas** **c) Resultado con cinco pasadas**

Figura 5.2. Comparación del resultado de la segmentación

Se ve bastante claro que al pasar más veces por el modelo se obtiene una mejor segmentación, la mesa se detecta más, al igual que la silla y el sofá. Además, en la Figura 5.2.a. en el sofá aparecen muchos píxeles de diferentes colores, detectando objetos que no aparecen en la imagen original mientras que en las Figuras 5.2.b. y 5.2.c se corrige. Sin embargo, con tres pasadas el funcionamiento a tiempo real es un poco más fluido y el resultado bastante parecido a cinco pasadas.

Otro ejemplo se puede apreciar en la Figura 5.4, cuya imagen original es la Figura 5.3. Con una pasada la mesa no se detecta, a pesar de que se encuentra justo delante, mientras que con cinco pasadas sí que se detecta, con detalles adicionales como los platos.

Figura 5.3. Imagen de un comedor de Gazebo

a) Resultado con una pasada **b) Resultado con cinco pasadas**

Figura 5.4. Comparación del resultado de la segmentación en un comedor

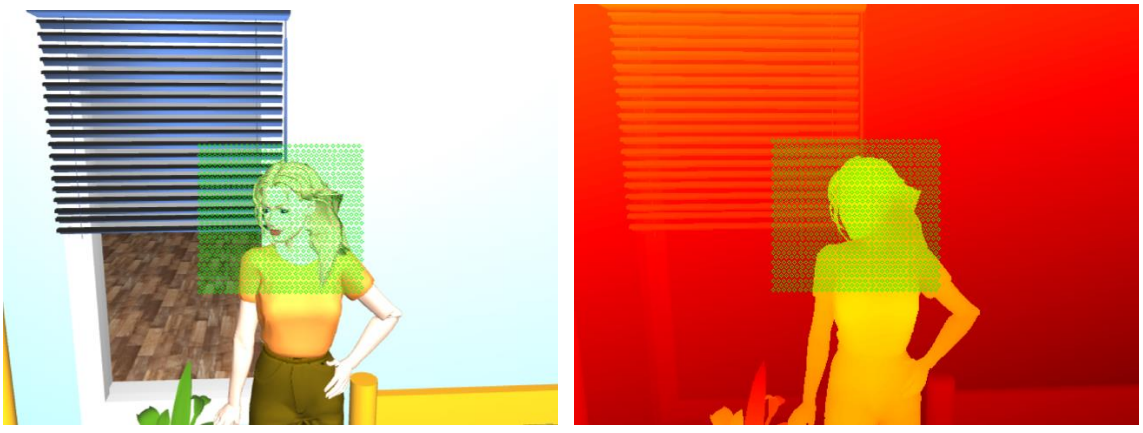
Finalmente se publica la imagen en los topics que se crearon al inicio, uno en color como las Figuras 5.2. o 5.4. para supervisión del nodo y otro en blanco y negro que se usa para resaltar objetos en el mapa de fosfenos.

Se comprueba que la red funciona correctamente y que puede funcionar a una velocidad fluida y apta para el manejo natural del mismo (a unos 10 FPS considerando el procesamiento de fosfenos y la visualización de las Oculus y movimiento de los ángulos). La segmentación es un poco más lenta (4 FPS), pero se espera que con los rápidos avances en deep learning y en las especificaciones de las tarjetas gráficas esto sea mucho más rápido en un futuro próximo. Se integra con RGBD-VIA, la herramienta con la que se realizan todas las transformaciones a fosfenos. Los tiempos de cómputo pueden observarse en la Tabla 5.1.

Sockets	Tiempo de inicialización	5s
	Tiempo de envío y recibo de datos	0.06s
	Tiempo total de bucle	0.09s
Segmentación de objetos	Tiempo de actualización con tres pasadas	0.25s
RGBD-VIA	Tiempo mínimo (se da en downsampling)	0.10s
	Tiempo máximo (se da en el detector de objetos y paredes)	0.4s

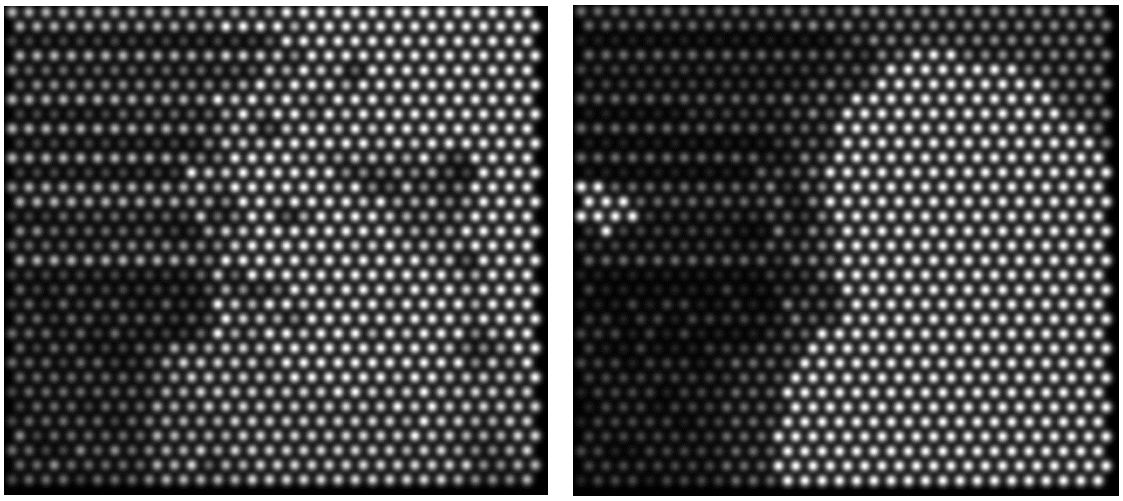
Tabla 5.1. Tiempos de cómputo

Se crea un nuevo modo de visualización en el que se resaltan mediante fosfenos de máxima luminosidad los objetos segmentados sobre la imagen base a la que la intensidad de los fosfenos se le ha reducido previamente a la mitad para que resalte más aún el objeto segmentado. Esta imagen puede ser cualquiera de las representaciones explicadas en el apartado 2.2.1, y se puede desactivar en cualquier momento pulsando una tecla para volver a observar las representaciones originales, al igual que pulsando diferentes teclas se puede cambiar de representación en cualquier instante. La idea es hacer de este método una herramienta de asistencia para que en momentos puntuales se pueda activar el reconocimiento de objetos (eligiendo qué objetos mostrar) y así mejorar su calidad de vida. En el ejemplo de a continuación se parte de la Figura 5.5. y se obtienen los resultados de las Figuras 5.6. y 5.7.



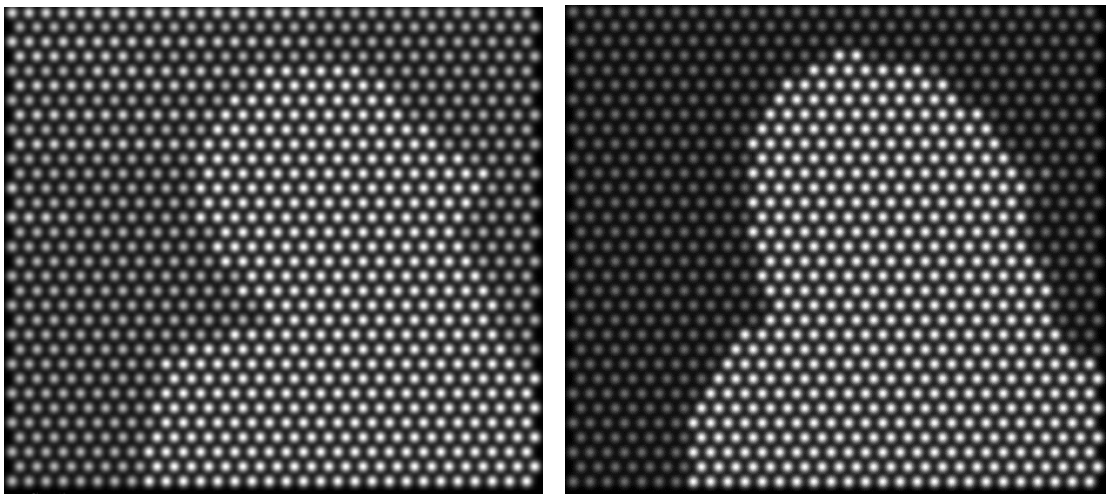
a) Imagen a color **b) Imagen de profundidad**

Figura 5.5. Imagen original de Gazebo



a) Representación Downsampling **b) Donwsampling y segmentación unidas**

Figura 5.6. Visualización de Downsampling con y sin segmentación para un campo de visión de 20º

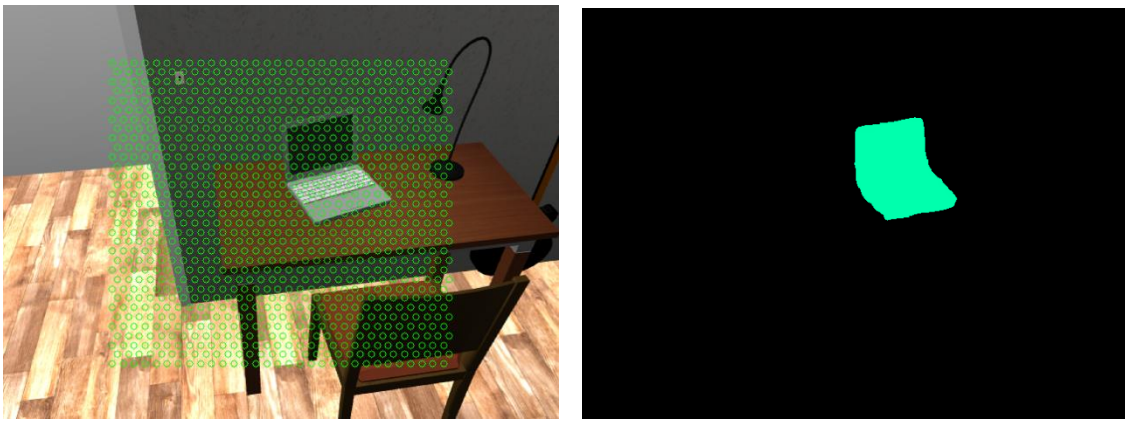


a) Representación modo de profundidad **b) Modo de profundidad y segmentación unidas**

Figura 5.7. Visualización de modo de profundidad con y sin segmentación para un campo de visión de 20º

Como se puede apreciar en las Figuras 5.6.b. y 5.7.b. se ha añadido la segmentación, con píxeles luminosos que representan una persona, mientras que el resto de píxeles aparecen más apagados que en sus representaciones originales (Figuras 5.6.a. y 5.7.a.). En el caso del modo de profundidad, en su representación original ya se distingue a la persona, pero al añadir la segmentación, como se reduce de intensidad todos los píxeles que no pertenecen a objetos segmentados, se percibe de una forma más clara a la persona.

Otro ejemplo se puede observar en la Figura 5.9, donde solo se ha decidido localizar el objeto ordenador (segmentación en Figura 5.8.b.):



a) Imagen de un ordenador de la cámara Gazebo b) Segmentación del objeto ordenador

Figura 5.8. Imagen de referencia y segmentación del objeto portátil

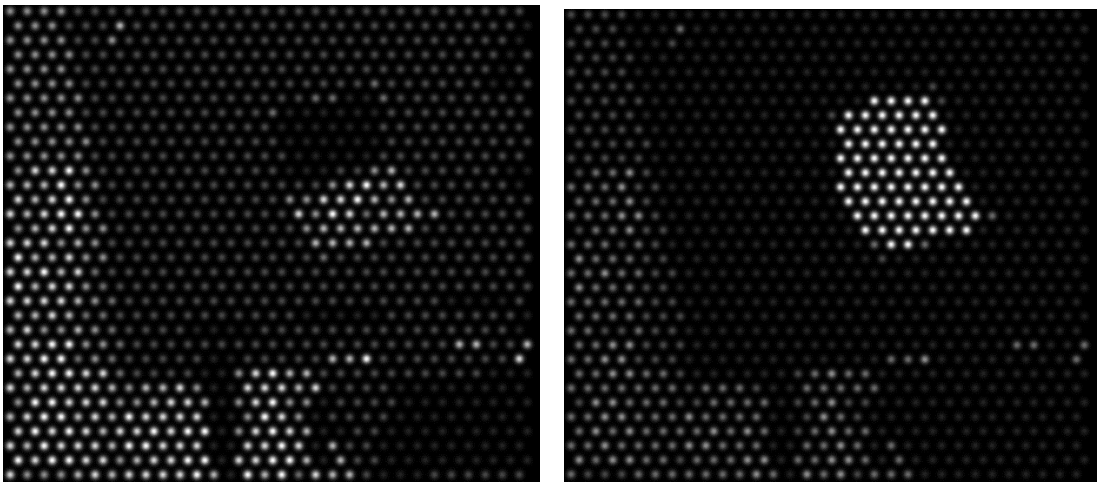
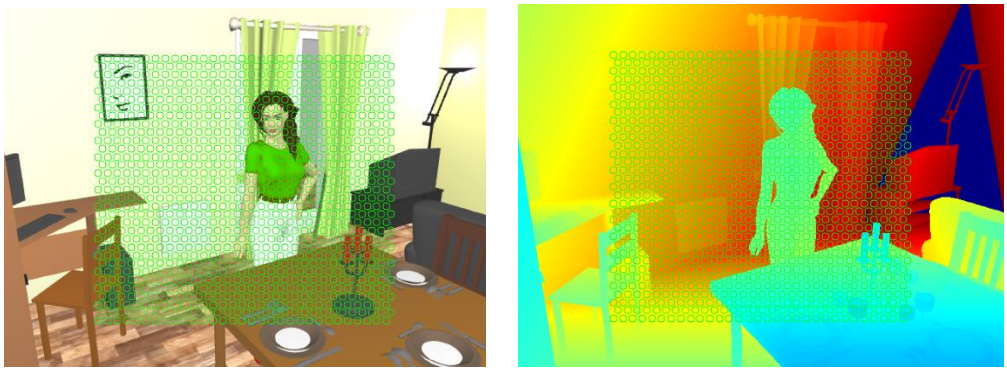


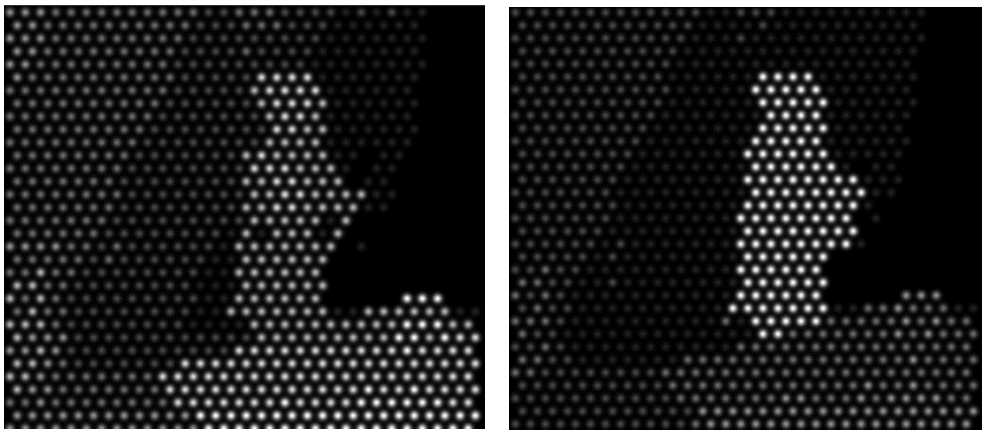
Figura 5.9. Visualización de Downsampling para el objeto ordenador con y sin segmentación para un campo de visión de 40º

Y si se desea únicamente detectar personas, en las Figuras 5.10. 5.11. y 5.12 se muestra un ejemplo para diferentes representaciones con un campo de visión más amplio:



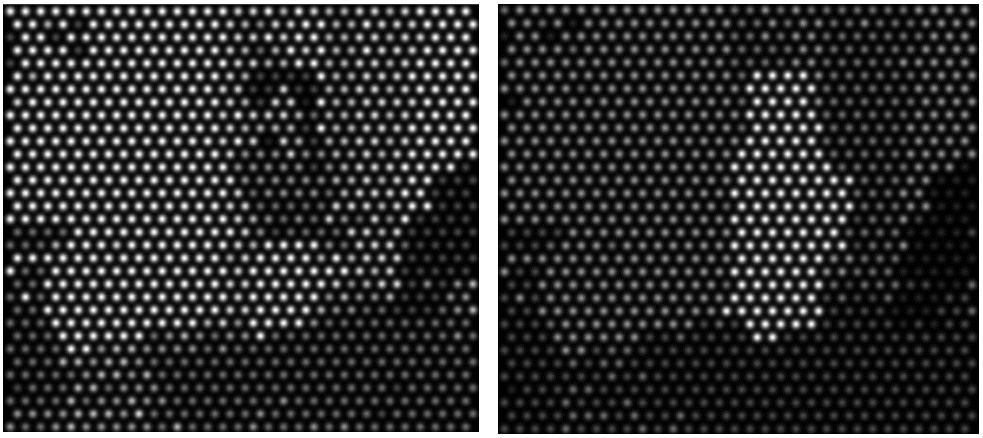
a) Imagen de color de la cámara de Gazebo b) Imagen de profundidad de la cámara de Gazebo

Figura 5.10. Imagen de referencia para la detección de personas



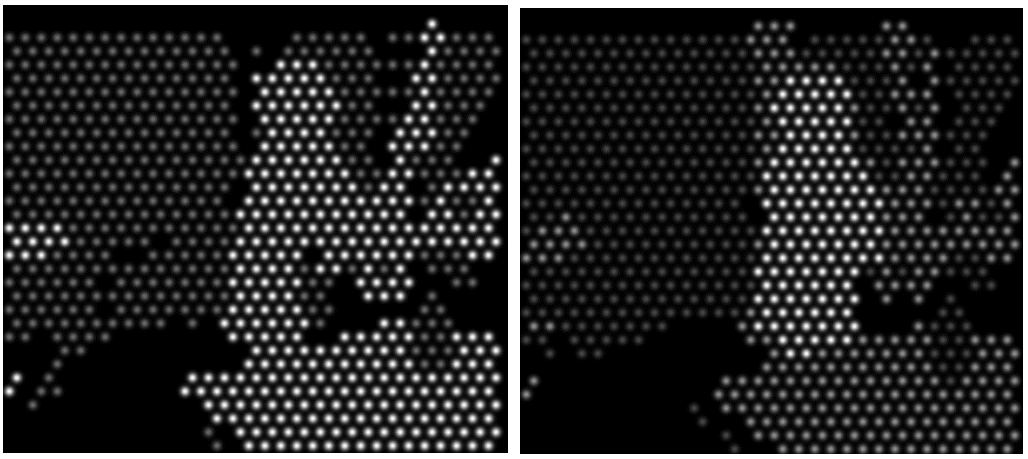
a) Representación del modo de profundidad **b) Modo de profundidad y segmentación unidas**

Figura 5.11. Visualización del modo de profundidad para la detección de personas con y sin segmentación para un campo de visión de 40º



a) Representación Downsampling **b) Donwsampling y segmentación unidas**

Figura 5.12. Visualización de Downsampling para la detección de personas con y sin segmentación para un campo de visión de 40º



a) Representación método de detección de obstáculos y paredes **b) Método de detección de obstáculos y paredes y segmentación unidas**

Figura 5.13. Visualización de método de detección de obstáculos y paredes para la detección de personas con y sin segmentación para un campo de visión de 40º

Como se puede observar en los tres ejemplos, a pesar de que en las representaciones originales hay alguna que detecta a la persona bastante bien (Figura 5.11.a), hay otros como el método de detección de obstáculos y paredes que no se llega a saber con exactitud. Además, en las representaciones mencionadas en la sección 2.2. no existe ningún tipo de razonamiento semántico y requiere de una interpretación de la escena con la prótesis, la cual tiene muchas limitaciones. Al introducir la segmentación de objetos con los objetos más interesantes, como por ejemplo para que detecte ordenadores, permite saber unívocamente que eso es un ordenador.

Capítulo 6

Conclusiones

Para concluir el trabajo se van a resumir los diferentes aspectos desarrollados en la memoria. Inicialmente se introduce el estado del arte actual de las prótesis, la motivación que ha dado lugar a este proyecto y los objetivos planteados, así como las herramientas usadas.

Una vez concluida la tarea, se introduce la representación mediante mapas de fosfenos y las representaciones ya implementadas en RGBD-VIA que se pueden visualizar.

Una vez comprendido cómo se obtiene la imagen final del visor de las gafas Oculus Rift DK2 el siguiente paso ha sido la implementación en Gazebo los ángulos *roll-pitch-yaw* para que la experiencia sea lo más inmersiva posible.

Para visualizar la imagen en el visor de las gafas se ha explicado la comunicación por *sockets* entre equipos y las diferencias entre cada uno. Adicionalmente se explica cómo se envía la información internamente en ROS.

Por último, se ha introducido un modelo pre-entrenado de segmentación de objetos para poder introducir una nueva representación de mapa de fosfenos.

Este trabajo tiene como finalidad el crear un SPV en tiempo real que permita la evaluación los métodos de representación de fosfenos, y el impacto que tiene poder explorar completamente el entorno en el reconocimiento de objetos. A diferencia de otros simuladores, éste es capaz de representar diferentes entornos virtuales de interiores, con

diferentes estancias, con la posibilidad de explorarlos completamente sin necesidad de desplazarse en la realidad.

Se ha logrado una implementación en tiempo real satisfactoria tal y como se puede observar en la Tabla 5.1, y además con posibilidad de poder ser extrapolado, es decir, podría crearse un nodo que leyera la información de una cámara real en vez de usar la de Gazebo y poder seguir funcionando todo, la comunicación por socket, la visualización de los fosfenos y la segmentación de objetos.

Como trabajo futuro se considera apropiado la realización de pruebas con usuarios para poder evaluar de forma efectiva la importancia de una exploración completa e inmersiva del entorno para reconocimiento de objetos, estancias y navegación por el mismo. Además de comparar los resultados obtenidos con y sin el uso del nuevo modo de visualización introducido con segmentación de objetos. Adicionalmente se podrá evaluar que objetos resaltar y cuántos para mejorar dichos resultados ya que, si se detectasen muchos, al visualizar en blanco los fosfenos pertenecientes a dichos objetos, se solaparían entre sí y no se distinguiría la forma de estos. También queda pendiente realizar experimentos en un entorno real mediante la lectura de una cámara RGB-D.

Para terminar, añadir que todos los objetivos fijados inicialmente han sido cumplidos y el terminar este trabajo ha sido satisfactorio a pesar de alguna dificultad en el proceso explicada más detalladamente en el Anexo I. La expectativa final es que este trabajo sirva como apoyo para continuar la investigación y ayude a mejorar la calidad de vida de los pacientes con estas enfermedades visuales.

Anexo I . Fases del proyecto y duración

En este Anexo se especificará cómo se ha desarrollado el proyecto paso por paso, el tiempo que se ha empleado en cada parte y los problemas presentados.

FASES	TIEMPO	OBSERVACIONES
Aprendizaje del uso de Sockets e implementación en Windows	Enero-Febrero	Complicaciones en la comprensión del funcionamiento de los Sockets y modificación para enviar diferentes tipos de datos y leerlos.
Realización de los tutoriales de ROS	Febrero-mitad Marzo	
Implementación del Socket en Ubuntu	Mitad Marzo – mitad Abril	Complicaciones en la interconexión de equipos y en el envío de información por topics.
Adecuación de funciones de fosfenos, comprobación de tiempos de computo	Mitad Abril – finales Abril	Diferentes versiones de las funciones en los equipos que impiden la correcta visualización
Implementación de los ángulos roll-pitch-yaw en Gazebo	Abril-Mayo	Complicaciones en el retardo del robot en responder.
Implementación de la segmentación semántica	Mayo-Junio	Complicaciones en el entendimiento de la red y en su modificación para usar topics

Figura A.1. Cronograma del proyecto

Como se puede observar en la Figura A.1. la primera fase fue aprender a utilizar los *socket*, mediante un ejemplo en Windows [57], donde se enviaban y recibían mensajes dentro del propio ordenador. Para poder implementar correctamente tanto el lado del cliente del Socket como el del servidor, se usaron programas adicionales que permitían poder "visualizar" lo que ocurre en la comunicación del socket.

Primero, se usó el programa Packet Sender que sirve como terminal para que al probar el programa de cliente se pudiera comprobar su correcto funcionamiento:

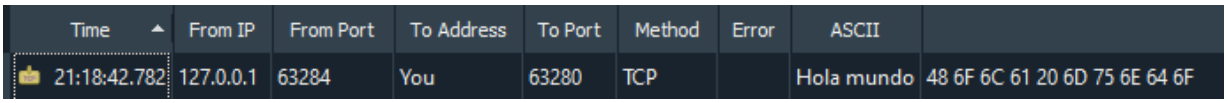


Figura A.2. Packet Sender

En la Figura A.2. se puede observar que el programa cliente ha enviado un mensaje de "Hola mundo" al puerto que utiliza el programa Packet Sender, si se utilizase otro distinto se enviaría el mensaje, pero no se podría saber si se ha enviado correctamente.

Una vez que se verifica que el programa cliente funciona correctamente, se procedió a hacer lo mismo con el programa servidor gracias a la herramienta SmartSniff (Figura A.3.) que permite visualizar la información que se envía, desde qué IP y puerto:

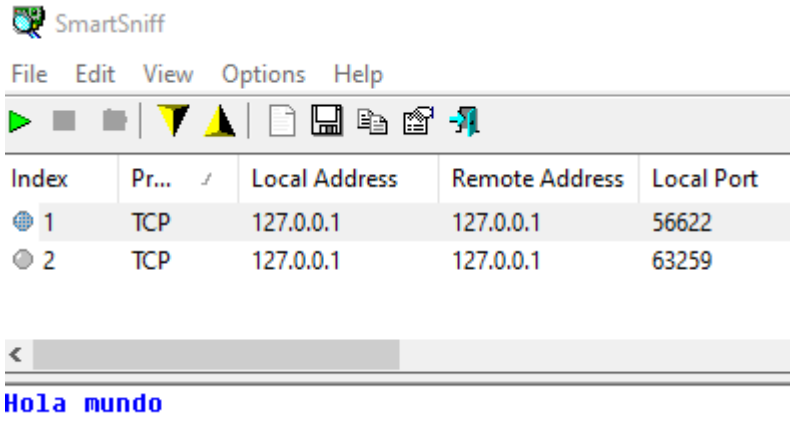


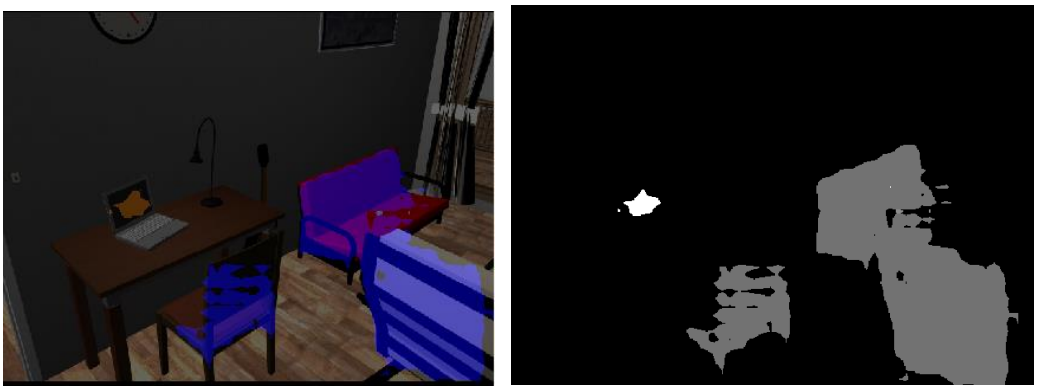
Figura A.3. SmartSniff

Una vez comprobado que los programas del tutorial funcionan correctamente, se modificaron para poder mandar otro tipo de datos diferentes para poder enviar la orientación de las gafas en cada momento y recibir la información de los fosfenos.

Una vez realizada esa tarea, se procedió a realizar tutoriales en el servidor Ubuntu sobre el uso de ROS [58], el uso de nodos, topics, etc. De igual manera que en Windows, se implementó un socket de prueba para mandar mensajes dentro de puertos del propio equipo y una vez comprobado que funcionaba el ejemplo se modificó el archivo del servidor para suscribirse y publicar en los topics adecuados y mandar y recibir información de los fosfenos y de los ángulos respectivamente a través del socket.

Se adecuaron las versiones de las funciones de los fosfenos de ambos equipos para conseguir la correcta visualización ya que, al calcular la posición, tamaño y número de fosfenos de forma diferente, la información se mandaba por el socket de forma correcta pero no se visualizaba bien. También se implementaron los ángulos de *roll-pitch-yaw* en Gazebo ya que solo estaba implementado uno y se modificó el movimiento para que fuera instantáneo y no tuviera ningún tipo de retraso como sí ocurre con los robots. Se tomaron tiempos de cómputo para corroborar si había algún cuello de botella y si se podía usar en tiempo real o por el contrario era necesario realizar cambios.

Por último, se implementaron dos nodos de segmentación antes de dar con el definitivo. El primero [59], consistía en una red pre-entrenada de modelos de DeepLab en tensorflow, sin embargo aunque parecía prometedor, y a pesar de que los veinte objetos que detectaba los localizaba muy bien, las mesas y estanterías por ejemplo no eran segmentadas (Figura A.4.), además esta red sí que localizaba varios objetos de exterior como autobuses o barcos por lo que no era adecuada para el simulador.



a) Imagen original y segmentada combinadas **b) imagen segmentada**

Figura A.4. Segmentación Deeplab

Otro *nodo* que se implementó, en pytorch [60], servía para la navegación de un vehículo, y solo detectaba el suelo de la carretera. En la Figura A.5. se ha introducido en Gazebo una ambulancia para que el modelo lo detecte y extraiga lo que sería la carretera. Es evidente que tampoco sirve, ya que los SPV están principalmente orientados para interiores, sin embargo, conocer su funcionamiento en pytorch sirvió para implementar el último y definitivo nodo. Un nodo con un modelo de red pre-entrenada que permitía la segmentación de objetos para introducir otra representación de mapa de fosfenos.

a) Imagen original **b) imagen segmentada**

Figura A.5. Segmentación de navegación de vehículo

Anexo II . Funcionamiento de los sockets TCP/IP

Para poder llevar a cabo la comunicación entre el ordenador con sistema operativo Windows, en el que se conectan las gafas de realidad virtual, y el ordenador con sistema operativo Ubuntu, que tiene el simulador, se realiza mediante sockets.

La programación de sockets es una forma de conectar dos nodos en una red para comunicarse entre sí [61]. Uno de los terminales es el cliente (el que inicia la conexión, en este caso el equipo Windows) y otro es el servidor (el que hace la labor de oyente al inicio, aunque una vez establecida, puede enviar información al igual que recibirla)

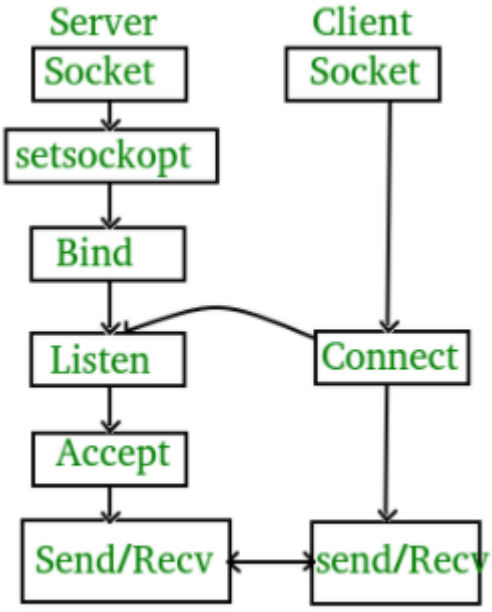


Figura A.6. Esquema del funcionamiento de un socket [61]

El comportamiento del cliente y del servidor es diferente, por eso a continuación se especifican los pasos para crear cada uno para una conexión TCP/IP.

En el lado del cliente [57], se inicializa *Winsock*, gracias a la función *WSAStartup* que inicializa el uso de WS2_32.dll. Se necesita declarar un objeto de tipo *addrinfo*, de tal forma que se use el protocolo TCP. Con la función *getaddrinfo* que solicita como argumentos el puerto y la dirección IP, se comprueba que esté disponible para conectarse. Con la función socket se crea el socket con las especificaciones deseadas, el protocolo, la familia de direcciones, etc.

Una vez creado el socket, es necesario realizar la conexión, con la función connect y los parámetros obtenidos de *getaddrinfo* y se procede a enviar y recibir información con las funciones *send* y *recv*. Por último, cuando termina de enviar o recibir datos, se desconecta del *socket* con la función *shutdown* y se cierra el socket con la función *closesocket*.

Por el caso del servidor [61][57], se crea de manera algo diferente, puesto que se va a ejecutar en Linux. Se crea el socket determinando la familia de direcciones que usa (IPv4) y el protocolo (TCP). Para que un servidor acepte conexiones de cliente, debe estar vinculado a una dirección de red dentro del sistema.

Una vez vinculado, el servidor debe escuchar para las solicitudes de conexión entrantes gracias a la función listen y aceptar dicha conexión utilizando un *socket* temporal para ello.

Tras la vinculación y aceptación de la conexión, es cuando se puede enviar y recibir información entre ambos terminales usando las funciones *send* y *recv* y cuando ya no se necesite más, se desconecta y se apaga el socket.

Anexo III . Conceptos sobre ROS

En este Anexo se especificarán ciertos conceptos sobre ROS.

AIII.1 Nodos

Los *nodos* son programas ejecutables que realizan funciones concretas, cada *nodo* sirve para un propósito concreto [62]. En este trabajo por ejemplo existe un *nodo* que ejecuta la segmentación semántica de objetos, otro que se encarga de la visualización de fosfenos, otro que recibe la información del *socket,* etc.

AIII.2 Topics

Los nodos se comunican mediante *topics*, enviando y recibiendo mensajes. Cuando se inicializa un topic se define además el tipo de mensaje que se puede publicar en él, podrá ser un mensaje de tipo texto, un array de enteros, etc.

Los nodos pueden enviar mensajes a través de estos topics o recibir mensajes siempre y cuando se suscriban al topic del que se desea recibir información. La comunicación en los topics es siempre unidireccional y asíncrona.

AIII.3 Mensajes

Son el tipo de estructura de datos usados en las comunicaciones internas entre los nodos [62]. Pueden ser de muchos tipos: integer, Float, boolean, etc. incluso tipos más complejos como sensor_msgs/Image.msg, que es una estructura que contiene una imagen sin comprimir.

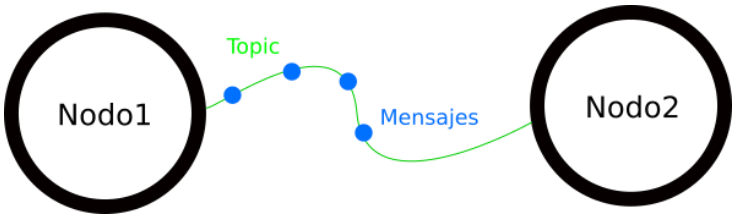


Figura A.7. Representación del funcionamiento de nodos y topics [63]

AIII.4. Fichero URDF

El formato de descripción robótica unificada (URDF) es un formato de archivo XML utilizado en ROS para describir todos los elementos de un robot [64]. Permite especificar las propiedades cinemáticas y dinámicas de un robot.

AIII.5 Joints

El elemento *joint* describe la cinemática y la dinámica de la articulación y también especifica los límites de seguridad [65].

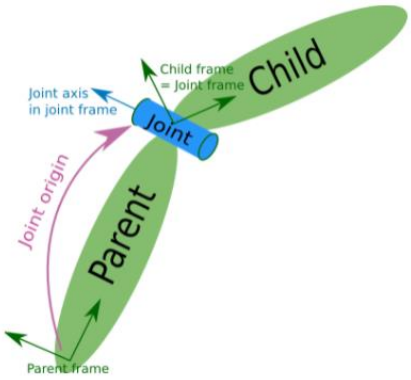


Figura A.8. Esquema del funcionamiento de una joint [65]

AIII.6 Links

Es un elemento de enlace que describe un cuerpo rígido con inercia, características visuales y propiedades de colisión [66].

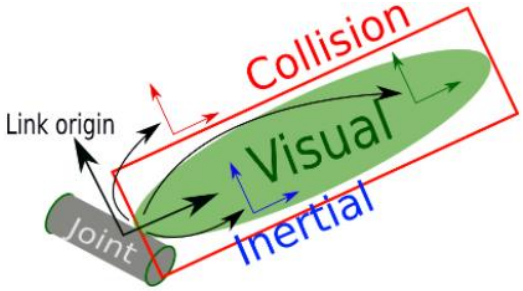


Figura A.9. Esquema del funcionamiento del fichero URDF [66]

Bibliografía

[1] R. R. A. Bourne *et al.*, "Magnitude, temporal trends, and projections of the global prevalence of blindness and distance and near vision impairment: a systematic review and meta-analysis," *Lancet Glob. Heal.*, vol. 5, no. 9, pp. e888–e897, Sep. 2017, doi: 10.1016/S2214-109X(17)30293-0.

[2] M. Santos Villafranca, "Simulador de protesis visual en entornos 360º con gafas de realidad virtual," *Zaragoza*. 2019.

[3] M. Sanchez-Garcia, R. Martinez-Cantin, and J. J. Guerrero, "Indoor Scenes Understanding for Visual Prosthesis with Fully Convolutional Networks," 2013.

[4] G. S. Brindley and W. S. Lewin, "The sensations produced by electrical stimulation of the visual cortex," *J. Physiol.*, vol. 196, no. 2, pp. 479–493, May 1968, doi: 10.1113/jphysiol.1968.sp008519.

[5] G. Dagnelie, *Visual prosthetics : physiology, bioengineering and rehabilitation*. Springer, 2011.

[6] W. L. D. Lui, D. Browne, L. Kleeman, T. Drummond, and W. H. Li, "Transformative Reality: Improving bionic vision with robotic sensing," *Proc. Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. EMBS*, pp. 304–307, 2012, doi: 10.1109/EMBC.2012.6345929.

[7] K. Cha, K. W. Horch, and R. A. Normann, "Mobility performance with a pixelized vision system," *Vision Res.*, vol. 32, no. 7, 1992, doi: 10.1016/0042-6989(92)90229-C.

[8] J. J. van Rheede, C. Kennard, and S. L. Hicks, "Simulating prosthetic vision: Optimizing the information content of a limited visual display," *J. Vis.*, vol. 10, no. 14, pp. 1–14, 2010, doi: 10.1167/10.14.1.

[9] S. C. Chen, L. E. Hallum, G. J. Suaning, and N. H. Lovell, “A quantitative analysis of head movement behaviour during visual acuity assessment under prosthetic vision simulation,” *J. Neural Eng.*, vol. 4, no. 1, 2007, doi: 10.1088/1741-2560/4/1/S13.

[10] S. C. Chen, G. J. Suaning, J. W. Morley, and N. H. Lovell, “Simulating prosthetic vision: I. Visual models of phosphenes,” *Vision Research*, vol. 49, no. 12. pp. 1493–1506, Jun. 2009, doi: 10.1016/j.visres.2009.02.003.

[11] S. C. Chen, G. J. Suaning, J. W. Morley, and N. H. Lovell, “Simulating prosthetic vision: II. Measuring functional capacity,” *Vision Research*, vol. 49, no. 19. Pergamon, pp. 2329–2343, Sep. 30, 2009, doi: 10.1016/j.visres.2009.07.003.

[12] M. Leo, A. Furnari, G. G. Medioni, M. Trivedi, and G. M. Farinella, “Deep learning for assistive computer vision,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2019, vol. 11134 LNCS, pp. 3–14, doi: 10.1007/978-3-030-11024-6_1.

[13] A. Badías Herbera, “Simulación de prótesis visual con sensor RGB-D.” pp. 2014–2015, 2015.

[14] L. Montano Oliván, “Guiado de personas ciegas mediante un simulador de visión protésica en entornos virtuales,” *Zaragoza*. pp. 1–107, 2015.

[15] K. Cha, D. K. Boman, K. W. Horch, and R. A. Normann, “Reading speed with a pixelized vision system,” *J. Opt. Soc. Am. A*, vol. 9, no. 5, p. 673, 1992, doi: 10.1364/josaa.9.000673.

[16] R. W. Thompson, G. D. Barnett, M. S. Humayun, and G. Dagnelie, “Facial Recognition Using Simulated Prosthetic Pixelized Vision,” *Investig. Ophthalmol. Vis. Sci.*, vol. 44, no. 11, pp. 5035–5042, 2003, doi: 10.1167/iovs.03-0341.

[17] G. Dagnelie, P. Keane, V. Narla, L. Yang, J. Weiland, and M. Humayun, “Real and virtual mobility performance in simulated prosthetic vision,” *J. Neural Eng.*, vol. 4, no. 1, Mar. 2007, doi: 10.1088/1741-2560/4/1/S11.

[18] J. Horace, M. Collette, K. Lindsay, W. Lui, and D. Lik, “Psychophysics testing of bionic vision image processing algorithms using an fpga hatpack Horace Josh , Collette Mann , Lindsay Kleeman and Wen Lik Dennis Lui Monash Vision Group and Monash University , Clayton , Australia,” *Ieee*, pp. 1550–1554, 2013.

[19] V. Vergnieux, M. J. M. Macé, and C. Jouffrais, “Wayfinding with simulated prosthetic vision: Performance comparison with regular and structure-enhanced renderings,” *2014 36th Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. EMBC 2014*, pp. 2585–2588, 2014, doi: 10.1109/EMBC.2014.6944151.

[20] M. P. H. Zapf, M. Y. Boon, P. B. Matteucci, N. H. Lovell, and G. J. Suaning, “Towards an assistive peripheral visual prosthesis for long-term treatment of retinitis pigmentosa: Evaluating mobility performance in immersive simulations,” *J. Neural Eng.*, vol. 12, no. 3, p. 36001, 2015, doi: 10.1088/1741-2560/12/3/036001.

[21] M. P. H. Zapf, M. Y. Boon, N. H. Lovell, and G. J. Suaning, “Assistive peripheral phosphene arrays deliver advantages in obstacle avoidance in simulated end-stage retinitis pigmentosa: A virtual-reality study,” *J. Neural Eng.*, vol. 13, no. 2, p. 0, 2016, doi: 10.1088/1741-2560/13/2/026022.

[22] Y. Zhao, X. Geng, Q. Li, G. Jiang, Y. Gu, and X. Lv, “Recognition of a Virtual Scene via Simulated Prosthetic Vision,” *Front. Bioeng. Biotechnol.*, vol. 5, Oct. 2017, doi: 10.3389/fbioe.2017.00058.

[23] M. Guerrero Viu, “Detección de personas para simulación de prótesis visual.” 2016.

[24] “Oculus Rift: visor de realidad virtual para ordenadores optimizados para la realidad virtual | Oculus.” https://www.oculus.com/rift/#oui-csl-rift-games=robo-recall (accessed May 26, 2019).

[25] “Open Source Release of Rift DK2 | Oculus.” https://developer.oculus.com/blog/open-source-release-of-rift-dk2/ (accessed May 25, 2019).

[26] “Oculus Rift Development Kit 2 Teardown - iFixit.” https://es.ifixit.com/Desmontaje/Oculus+Rift+Development+Kit+2+Teardown/27613 (accessed May 26, 2019).

[27] “Samsung Galaxy Note 3 specs | Android Central.” https://www.androidcentral.com/samsung-galaxy-note-3-specs (accessed May 26, 2019).

[28] “Introducción a la visión por computador: desarrollo de aplicaciones con OpenCV | edX.” https://www.edx.org/es/course/introduccion-a-la-vision-por-computador-desarrollo?index=spanish_product&queryID=9050cdeef7a5eb4c4066d159f9baf2ac&position=1 (accessed Jun. 20, 2021).

[29] “Robot Operating System - Wikipedia, la enciclopedia libre.” https://es.wikipedia.org/wiki/Robot_Operating_System (accessed May 26, 2021).

[30] A. Perez-Yus, J. Bermudez-Cameo, G. Lopez-Nicolas, and J. J. Guerrero, “Depth and motion cues with phosphene patterns for prosthetic vision,” pp. 1516–1525.

[31] “Capítulo 3. Entorno de simulación ROs/Gazebo.” http://bibing.us.es/proyectos/abreproy/5139/fichero/PFC-+Por+capítulos%252F3-Entorno+de+simulación.pdf (accessed May 26, 2021).

[32] L. C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, “DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, pp. 834–848, Apr. 2018, doi: 10.1109/TPAMI.2017.2699184.

[33] F. Yu and V. Koltun, “Multi-Scale context aggregation by dilated convolutions.”

[34] M. Sanchez-Garcia, R. Martinez-Gantin, and J. J. Guerrero, “Semantic and structural image segmentation for prosthetic vision,” 2019.

[35] “Prótesis retinales: ¿Una segunda oportunidad para los ojos? – Blog Oftal.” https://oftalmologoaldia.com/blog/2019/05/09/protesis-retinales-una-segunda-oportunidad-para-los-ojos/ (accessed Jun. 08, 2021).

[36] J. Weiland and M. Humayun, “Retinal prosthesis,” in *Neural Engineering: Second Edition*, Springer US, 2013, pp. 635–655.

[37] J. Hirsch and C. A. Curcio, “The spatial resolution capacity of human foveal retina,” *Vision Res.*, vol. 29, no. 9, pp. 1095–1101, 1989, doi: 10.1016/0042-6989(89)90058-8.

[38] J. D. Weiland, A. K. Cho, and M. S. Humayun, “Retinal prostheses: Current clinical results and future needs,” *Ophthalmology*, vol. 118, no. 11. Elsevier, pp. 2227–2237, Nov. 01, 2011, doi: 10.1016/j.ophtha.2011.08.042.

[39] E. Zrenner *et al.*, “Subretinal microelectrode arrays allow blind retinitis pigmentosa patients to recognize letters and combine them to words,” 2009, doi: 10.1109/BMEI.2009.5305315.

[40] H. Meffin, “What limits spatial perception with retinal implants? NeuroEngineering Laboratory & Centre for Neural Engineering , The University of Melbourne,” pp. 1545–1549, 2013.

[41] H. C. Stronks and G. Dagnelie, “The functional performance of the Argus II retinal prosthesis,” doi: 10.1586/17434440.2014.862494.

[42] “Estimulación del nervio óptico para dar visión a los ciegos | Asociación Mácula Retina.” https://www.macula-retina.es/estimulacion-del-nervio-optico-para-dar-vision-a-los-ciegos/ (accessed Jun. 18, 2021).

[43] C. Veraart *et al.*, “Visual sensations produced by optic nerve stimulation using an implanted self-sizing spiral cuff electrode,” *Brain Res.*, vol. 813, no. 1, pp. 181–186, Nov. 1998, doi: 10.1016/S0006-8993(98)00977-9.

[44] J. Delbeke, M. Oozeer, and C. Veraart, “Position, size and luminosity of phosphenes generated by direct optic nerve stimulation,” *Vision Res.*, vol. 43, no. 9, pp. 1091–1102, 2003, doi: 10.1016/S0042-6989(03)00013-0.

[45] M. E. Brelén, F. Duret, B. Gérard, J. Delbeke, and C. Veraart, “Creating a meaningful visual perception in blind volunteers by optic nerve stimulation,” in *Journal of Neural Engineering*, Mar. 2005, vol. 2, no. 1, doi: 10.1088/1741-2560/2/1/004.

[46] F. Duret, M. E. Brelén, V. Lambert, B. Gérard, J. Delbeke, and C. Veraart, "Object localization, discrimination, and grasping with the optic nerve visual prosthesis.," *Restor. Neurol. Neurosci.*, vol. 24, no. 1, pp. 31–40, Jan. 2006, [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/16518026.

[47] M. S. Humayun *et al.*, "Visual perception in a blind subject with a chronic microelectronic retinal prosthesis," *Vision Res.*, vol. 43, no. 24, pp. 2573–2581, Nov. 2003, doi: 10.1016/S0042-6989(03)00457-7.

[48] S. C. Chen, N. H. Lovell, and G. J. Suaning, "Effect on prosthetic vision visual acuity by filtering schemes, filter cut-off frequency and phosphene matrix: A virtual reality simulation," *Annu. Int. Conf. IEEE Eng. Med. Biol. - Proc.*, vol. 26 VI, pp. 4201–4204, 2004, doi: 10.1109/iembs.2004.1404172.

[49] G. Dagnelie, "Visual prosthetics 2006: Assessment and expectations," *Expert Review of Medical Devices*, vol. 3, no. 3. pp. 315–325, May 2006, doi: 10.1586/17434440.3.3.315.

[50] C. McCarthy, J. G. Walker, P. Lieby, A. Scott, and N. Barnes, "Mobility and low contrast trip hazard avoidance using augmented depth," *J. Neural Eng.*, vol. 12, no. 1, Feb. 2015, doi: 10.1088/1741-2560/12/1/016003.

[51] "turtlebot_simulator - ROS Wiki." http://wiki.ros.org/turtlebot_simulator (accessed Jun. 14, 2021).

[52] "gazebo set joint angles by ROS - ROS Answers: Open Source Q&A Forum." https://answers.ros.org/question/233059/gazebo-set-joint-angles-by-ros/ (accessed Jun. 04, 2021).

[53] "Sweet Home 3D." http://www.sweethome3d.com/es/ (accessed Jun. 05, 2021).

[54] "Socket de Internet - Wikipedia, la enciclopedia libre." https://es.wikipedia.org/wiki/Socket_de_Internet (accessed May 26, 2021).

[55] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba, "Scene Parsing through ADE20K Dataset." Accessed: Jun. 15, 2021. [Online]. Available: http://groups.csail.mit.edu/vision/datasets/ADE20K/.

[56] B. Zhou *et al.*, “Semantic Understanding of Scenes through the ADE20K Dataset.” Accessed: Jun. 15, 2021. [Online]. Available: http://groups.csail.mit.edu/vision/datasets/ADE20K.Pretrainedmodelsandcodeare releasedathttps://github.com/CSAILVision/semantic-segmentation-pytorch.

[57] “Getting Started with Winsock - Win32 apps | Microsoft Docs.” https://docs.microsoft.com/en-us/windows/win32/winsock/getting-started-with-winsock (accessed May 26, 2021).

[58] “ROS/Tutorials - ROS Wiki.” http://wiki.ros.org/ROS/Tutorials (accessed Jun. 01, 2021).

[59] “GitHub - ethz-asl/deeplab_ros: ROS Wrapper for DeepLab: Deep Labelling for Semantic Image Segmentation.” https://github.com/ethz-asl/deeplab_ros (accessed Jun. 09, 2021).

[60] “GitHub - dheera/ros-semantic-segmentation: ROS package for semantic segmentation.” https://github.com/dheera/ros-semantic-segmentation (accessed Jun. 09, 2021).

[61] “Socket Programming in C/C++ - GeeksforGeeks.” https://www.geeksforgeeks.org/socket-programming-cc/ (accessed May 26, 2021).

[62] “Conceptos básicos de ROS | Romerobots Blog.” https://jjromeromarras.wordpress.com/2014/08/27/conceptos-basicos-de-ros/ (accessed Jun. 07, 2021).

[63] “4 Nodos, Topics y Mensajes. Turtlesim - ROS TUTORIAL. Tutorial ROS en español.” http://rostutorial.com/4-nodos-topics-y-mensajes-turtlesim/ (accessed Jun. 07, 2021).

[64] “Gazebo : Tutorial : URDF in Gazebo.” http://gazebosim.org/tutorials?tut=ros_urdf (accessed Jun. 08, 2021).

[65] “urdf/XML/joint - ROS Wiki.” https://wiki.ros.org/urdf/XML/joint (accessed Jun. 08, 2021).

[66] “urdf/XML/link - ROS Wiki.” https://wiki.ros.org/urdf/XML/link (accessed Jun. 08, 2021).