
Implementación de un control de movimiento para dispositivos robóticos móviles utilizando identificación de gestos faciales y orientación de cabeza por medio de visión de computadora

Gerardo Andres Fuentes Bámaca



UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería



Implementación de un control de movimiento para dispositivos robóticos móviles utilizando identificación de gestos faciales y orientación de cabeza por medio de visión de computadora

Trabajo de graduación presentado por Gerardo Andres Fuentes Bámaca para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2024

Vo.Bo.:

(f) _____
Ph.D. Luis Alberto Rivera Estrada

Tribunal Examinador:

(f) _____
Ph.D. Luis Alberto Rivera Estrada

(f) _____
M. Sc. Miguel Enrique Zea

(f) _____
Ing. Kurt Kellner

Fecha de aprobación: Guatemala, 14 de junio de 2024.

Prefacio

Quiero empezar utilizando la conjugación del verbo querer, y así lo hice. La razón de esto, es porque hace poco escuché una frase que expresaba una idea de liberación: hacer lo que quiero y no lo que debo, sin embargo no recuerdo al autor dado que no leí la frase. Al inicio suena irresponsable, y sí, puede serlo dependiendo del enfoque, pero a su vez puede significar una increíble responsabilidad porque requiere una enorme transformación y compromiso individual el llegar a querer hacer lo que se debe hacer. Dicho esto quiero agradecer haber estado en esta posición y poder vivir hasta el día en que escribo esta sección de mi trabajo de graduación. La posición que hoy ocupo, es la de un ser humano que aunque muchos duden, busca el bienestar en general no solo propio, no solo de sus seres queridos, sino del mundo en general y si existe vida más allá de este planeta, también. Pero al mismo tiempo insignificante comparado con el mismo universo del que hablo. Sin embargo, no importa que tanta reflexión se realice acerca de nuestra pequeñez, hay algo que no podemos evitar, es que vivimos bajo la percepción de la realidad de un ser humano, por lo que sigo sintiendo miedo, amor, tristeza, alegría, enojo, etc. Dicho lo anterior, agradezco principalmente a mis seres progenitores, mi padre Rodolfo y mi madre Milbia, quienes a pesar de todo lo que ellos puedan decir que no nos dieron o que causó problemas en nuestras vidas, no fueron nada más que errores, errores de humanos que todos cometemos. Su intención siempre fue proveer el mejor bienestar posible, y se que han dado todo de sí para lograrlo, al ser humano soy moldeable, y lo que soy hoy, se los debo a ellos. En segundo lugar quiero agradecer por la existencia de mis hermanos, que de hecho también es gracias a mis padres. Mis hermanos desde una temprana edad me han motivado a perseguir el sueño de convertirme en un guía para las personas, aún cuando he fallado y he cometido errores de gran magnitud. Ellos me han demostrado que el arrepentimiento, el deseo genuino por mejorar y realizar acciones para recuperar el camino correcto, son posibles y si alguien lo logra, es digno de admirar. Lo que me lleva al siguiente grupo de personas, mis *abuelitos*, quienes me acogieron en su casa desde una muy temprana edad, y me enseñaron acerca de una perspectiva de vida que se está perdiendo actualmente, no necesariamente mala, ni buena, solo otra perspectiva. Y cuando me refiero a mis abuelos, no solo me refiero a los maternos, a ti también *abuelita*, como te solíamos llamar, porque como todas las personas en mi familia, hemos tenido dificultades iniciales en nuestra vida, pero seguimos adelante, con todo y nuestras fallas. Mis tíos, hermanos de mi madre, y sus familias, con o sin mucha interacción, me han enseñado muchos valores de la vida que quiero seguir y otros que no. A mis primos, quiénes han

tenido un rol un poco más secundario últimamente, pero que definitivamente marcaron de manera positiva mi infancia, y la hicieron especial a pesar de las dificultades del día a día. Asimismo, mi tío que me llevó al colegio durante muchos años, también marcó de manera positiva mi infancia. Hablando de infancia, agradezco mucho a mis maestras de cuarto primaria, una que en paz descance, por haber sido excelentes en mi proceso de transición de un estilo de educación a otro. También, agradeceré a todas las personas que en algún momento me desearon o me hicieron mal, porque me permitieron vivir la vida en sus muchas variedades, y así me permiten hoy tener un criterio mucho más completo acerca de la vida. Gracias a mis amigos del colegio y universidad, por hacerme sentir acompañado en este duro mundo. Y como olvidar a la tía de Estados Unidos que me apoyó en momentos delicados, económicamente hablando, de mi etapa universitaria, eternamente agradecido por la paz que le dió a mi familia cuando yo no fui capaz. A mi primera madrina de becas, quién fue la primera persona dentro de una institución becaria en ver y confiar en mi potencial, le prometo que voy a demostrar de que estoy hecho, aún me falta mucho por hacer. Por último, pero no menos importante, a la Fundación Juan Bautista Gutiérrez, en específico a Doña Isabelita, quien tuvo la decisión de confiar en mi y vió el potencial que tengo, suficiente como para ser acreedor de la beca, por la cual me estaré graduando de esta universidad. Al equipo de coordinadoras, que siempre fueron tan amables conmigo y también confiaron en mí en todo momento, también aprovecho a pedirles perdón por ciertas decisiones tomadas en el camino. Sin embargo, dichas decisiones me han hecho crecer en muchos aspectos y lo seguirán haciendo. En conclusión, gracias a todos por todo y en caso de no volver a vernos, buenos días, buenas tardes y buenas noches.

Índice

| | |
|-----------------------------------------------------------------------------------------------------------------------|-------------|
| Prefacio | IV |
| Lista de figuras | X |
| Lista de cuadros | XI |
| Resumen | XII |
| Abstract | XIII |
| 1. Introducción | 1 |
| 2. Antecedentes | 3 |
| 2.1. Control de gestos para su uso en automóviles | 3 |
| 2.2. Reconocimiento de gestos automático para interacciones humano-robot | 4 |
| 2.2.1. Modelo humano en 3D | 4 |
| 2.2.2. Extracción de características | 4 |
| 2.2.3. Fase de entrenamiento | 4 |
| 2.2.4. Fase de reconocimiento | 4 |
| 2.2.5. Resultados experimentales | 5 |
| 2.3. Sistema de reconocimiento de manos y rostro para personas ciegas | 5 |
| 2.4. Sistema de reconocimiento gestos de cabeza utilizando la cámara de gestos . . | 6 |
| 2.5. Control de una silla de ruedas usando una agrupación de k-medias adaptativas de las poses de la cabeza | 7 |
| 2.5.1. Calibración | 8 |
| 2.5.2. Rectificado | 8 |
| 2.5.3. Adaptación | 8 |
| 2.6. <i>Mediapipe</i> | 9 |
| 3. Justificación | 10 |
| 4. Objetivos | 11 |

| | |
|------------------------------------------------------------------------|-----------|
| 5. Alcance | 12 |
| 6. Marco teórico | 13 |
| 6.1. <i>Machine learning</i> | 13 |
| 6.1.1. Aprendizaje supervisado | 13 |
| 6.1.2. Aprendizaje no supervisado | 14 |
| 6.2. Redes Neuronales | 14 |
| 6.2.1. Estructura básica de las redes neuronales | 15 |
| 6.2.2. Redes neuronales convolucionales | 17 |
| 6.2.3. Redes neuronales de grafos | 18 |
| 6.3. Visión por computadora | 19 |
| 6.3.1. Procedimientos | 20 |
| 6.3.2. Hardware | 22 |
| 6.3.3. Software | 25 |
| 6.4. Protocolos de comunicación | 27 |
| 6.4.1. Serial | 27 |
| 6.4.2. <i>Bluetooth</i> | 28 |
| 6.5. Sistemas operativos en tiempo real | 28 |
| 7. Selección de hardware y software | 30 |
| 7.1. Lenguaje de programación | 30 |
| 7.2. Simulador | 30 |
| 7.3. Sensor óptico | 31 |
| 7.4. Agente robótico | 31 |
| 8. Bases de datos | 32 |
| 8.1. ICP-R (Grupo de modelos 1) | 32 |
| 8.1.1. Etiquetas para clasificación | 34 |
| 8.2. ICP-R y fotografías personales (Grupo de modelos 2 y 3) | 36 |
| 8.2.1. Ajuste | 37 |
| 8.3. Únicamente fotografías personales (Grupo de modelos 4) | 38 |
| 9. Algoritmos para visión por computadora | 39 |
| 9.1. Grupos de modelos: 1 a 3 | 39 |
| 9.1.1. Preprocesado de imágenes | 39 |
| 9.1.2. Entrenamiento de modelos de <i>machine learning</i> | 40 |
| 9.2. Grupo de modelos 4 | 53 |
| 9.2.1. Preprocesado de imágenes | 53 |
| 9.2.2. Entrenamiento de modelos de <i>machine learning</i> | 55 |
| 9.3. Programación multi-hilos | 62 |
| 9.4. Paro de emergencia | 62 |
| 10. Validación del sistema por medio de simulaciones | 63 |
| 10.1. Interfaz 1(Grupos de modelos: 1 a 3) | 63 |
| 10.1.1. Prueba en tiempo real | 63 |
| 10.1.2. Bloques de funciones | 64 |
| 10.2. Interfaz 2 (Grupo de modelos: 4) | 65 |
| 10.2.1. Captura de datos | 66 |
| 10.2.2. Manejo de datos | 66 |

| | |
|--------------------------------------------------------------------------|-----------|
| 10.3. Interfaz 3 | 67 |
| 10.3.1. Captura de datos con apoyo audiovisual | 68 |
| 10.4. Consola (Grupo de modelos: 1) | 69 |
| 10.5. <i>Turtle</i> | 69 |
| 10.5.1. Grupo de modelos: 2 | 69 |
| 10.5.2. Grupo de modelos: 4 | 70 |
| 10.6. <i>Webots</i> | 71 |
| 10.6.1. Grupo de modelos: 3 | 71 |
| 10.6.2. Grupo de modelos: 4 | 72 |
| 11. Validación del sistema con plataformas robóticas físicas | 73 |
| 11.1. Verificación de recepción de señales | 73 |
| 11.1.1. <i>PC-PYTHON</i> | 73 |
| 11.1.2. <i>ESP32/PROTOBOARD</i> | 73 |
| 11.2. Verificación de funcionamiento con agente robótico móvil | 74 |
| 11.2.1. Antirrebotes | 75 |
| 12. Conclusiones | 77 |
| 13. Recomendaciones | 79 |
| 14. Bibliografía | 81 |
| 15. Anexos | 84 |
| 15.1. Códigos | 84 |
| 15.2. Videos demostrativos | 84 |

Lista de figuras

| | | |
|-----|----------------------------------------------------------------------------------------------------------------------------------|----|
| 1. | Captura de datos para la trayectoria característica de dos gestos en un espacio cerrado. [2] | 5 |
| 2. | Arquitectura del sistema propuesto para reconocimiento de gestos de manos. [3] | 6 |
| 3. | Arquitectura del sistema propuesto para reconocimiento facial. [3] | 6 |
| 4. | Diagrama del sistema propuesto. [5] | 8 |
| 5. | Visualización de el mapeo de punto del rostro utilizando <i>Mediapipe</i> . [6] | 9 |
| 6. | Estructura gráfica de del funcionamiento del perceptrón [10] | 15 |
| 7. | Gráficas varias de funciones de activación [10] | 16 |
| 8. | Representación de una red multicapa para clasificación de colores[10] | 17 |
| 9. | Representación del proceso de las redes convolucionales [12] | 18 |
| 10. | Ejemplos de estructuras capaces de formar grafos[14] | 19 |
| 11. | Muestra de 15 fotografías del sujeto 12. [41] | 33 |
| 12. | Orden de guardado para cada serie de fotografías. (Generado por Gerardo Fuentes) | 33 |
| 13. | Clasificación de imágenes en 9 etiquetas. (Generado por Gerardo Fuentes) | 34 |
| 14. | Primera distribución de etiquetas. (Generado por Gerardo Fuentes) | 34 |
| 15. | Segunda distribución de etiquetas. (Generado por Gerardo Fuentes) | 35 |
| 16. | Clasificación de imágenes en 6 etiquetas. (Generado por Gerardo Fuentes) | 35 |
| 17. | Tercera distribución de etiquetas. (Generado por Gerardo Fuentes) | 36 |
| 18. | Cuarta distribución de etiquetas (Las posiciones se cuentan desde la esquina inferior izquierda). (Generado por Gerardo Fuentes) | 36 |
| 19. | Captura de fotografías simples. (Generado por Gerardo Fuentes) | 37 |
| 20. | Captura de fotografías con marcadores físicos. (Generado por Gerardo Fuentes) | 37 |
| 21. | Pérdida y exactitud del primer modelo con 9 clases. (Generado por Gerardo Fuentes) | 42 |
| 22. | Matriz de confusión del primer modelo con 9 clases. (Generado por Gerardo Fuentes) | 42 |
| 23. | Pérdida y exactitud del segundo modelo con 9 clases. (Generado por Gerardo Fuentes) | 43 |

| | |
|----------------------------------------------------------------------------------------------------------------------------------|----|
| 24. Matriz de confusión del segundo modelo con 9 clases. (Generado por Gerardo Fuentes) | 43 |
| 25. Pérdida y exactitud del primer modelo con 6 clases. (Generado por Gerardo Fuentes) | 44 |
| 26. Matriz de confusión del primer modelo con 6 clases. (Generado por Gerardo Fuentes) | 44 |
| 27. Pérdida y exactitud del segundo modelo con 6 clases. (Generado por Gerardo Fuentes) | 45 |
| 28. Matriz de confusión del segundo modelo con 6 clases. (Generado por Gerardo Fuentes) | 45 |
| 29. Pérdida y exactitud del modelo con 4 clases. (Generado por Gerardo Fuentes) | 46 |
| 30. Matriz de confusión del modelo con 4 clases. (Generado por Gerardo Fuentes) | 47 |
| 31. Pérdida y exactitud del modelo con 4 clases. (Generado por Gerardo Fuentes) | 48 |
| 32. Matriz de confusión del modelo con 4 clases. (Generado por Gerardo Fuentes) | 48 |
| 33. Pérdida y exactitud del modelo con 4 clases y aumento de imágenes de entrenamiento. (Generado por Gerardo Fuentes) | 49 |
| 34. Matriz de confusión del modelo con 4 clases y aumento de imágenes de entrenamiento. (Generado por Gerardo Fuentes) | 50 |
| 35. Pérdida y exactitud del modelo con 4 clases con filtro de bordes. (Generado por Gerardo Fuentes) | 51 |
| 36. Matriz de confusión del modelo con 4 clases con filtro de bordes. (Generado por Gerardo Fuentes) | 51 |
| 37. Pérdida y exactitud del modelo con 4 clases y marcadores faciales. (Generado por Gerardo Fuentes) | 52 |
| 38. Matriz de confusión del modelo con 4 clases y marcadores faciales. (Generado por Gerardo Fuentes) | 52 |
| 39. Despliegue de los puntos de referencia desde las funciones de <i>Mediapipe</i> . [6] | 53 |
| 40. Despliegue de los puntos de referencia desde la matriz obtenida. (Generado por Gerardo Fuentes) | 54 |
| 41. Pérdida y exactitud del modelo con 4 posiciones/gestos (sonrisa). (Generado por Gerardo Fuentes) | 56 |
| 42. Matriz de confusión del modelo con 4 posiciones/gestos (sonrisa). (Generado por Gerardo Fuentes) | 57 |
| 43. Pérdida y exactitud del modelo con 4 posiciones/gestos (boca abierta). (Generado por Gerardo Fuentes) | 58 |
| 44. Matriz de confusión del modelo con 4 posiciones/gestos (boca abierta). (Generado por Gerardo Fuentes) | 58 |
| 45. Pérdida y exactitud del modelo con 4 posiciones/gestos. (Generado por Gerardo Fuentes) | 59 |
| 46. Matriz de confusión del modelo con 4 posiciones/gestos. (Generado por Gerardo Fuentes) | 59 |
| 47. Pérdida y exactitud del modelo con 5 posiciones/gestos. (Generado por Gerardo Fuentes) | 60 |
| 48. Matriz de confusión del modelo con 5 posiciones/gestos. (Generado por Gerardo Fuentes) | 60 |
| 49. Pérdida y exactitud del modelo con 9 posiciones/gestos. (Generado por Gerardo Fuentes) | 61 |

| | | |
|-----|----------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 50. | Matriz de confusión del modelo con 9 posiciones/gestos. (Generado por Gerardo Fuentes) | 61 |
| 51. | Sección de captura de fotos en la interfaz. (Generado por Gerardo Fuentes) | 64 |
| 52. | Sección de manejo de datos en la interfaz. (Generado por Gerardo Fuentes) | 65 |
| 53. | Sección de pruebas en tiempo real con <i>Turtle</i> . (Generado por Gerardo Fuentes) | 65 |
| 54. | Sección de captura de fotos en la interfaz, versión 2. (Generado por Gerardo Fuentes) | 66 |
| 55. | Sección de manejo de datos en la interfaz, versión 2. (Generado por Gerardo Fuentes) | 67 |
| 56. | Interfaz gráfica con 3 métodos de traducción de predicción a comandos. (Generado por Gerardo Fuentes) | 68 |
| 57. | Posiciones de referencia para apoyo a usuarios. [42] | 69 |
| 58. | Prueba con trayectoria de un cuadrado en <i>Turtle</i> (modelos 2). (Generado por Gerardo Fuentes) | 70 |
| 59. | Prueba con trayectoria de un cuadrado en <i>Turtle</i> (modelos 4). (Generado por Gerardo Fuentes) | 71 |
| 60. | Prueba con trayectoria de un cuadrado en <i>Webots</i> (modelos 3). (Generado por Gerardo Fuentes) | 72 |
| 61. | Prueba con trayectoria de un cuadrado en <i>Webots</i> (modelos 4). (Generado por Gerardo Fuentes) | 72 |
| 62. | Configuración de pruebas para verificación de recepción de predicciones. (Generado por Gerardo Fuentes) | 74 |
| 63. | Placa para comunicación serial <i>ESP32-Pololu</i> (vista superior). (Generada por Gerardo Fuentes) | 75 |
| 64. | Placa para comunicación serial <i>ESP32-Pololu</i> (vista inferior). (Generada por Gerardo Fuentes) | 75 |
| 65. | Utilización del sistema de reconocimiento de gestos y posiciones para controlar el agente robótico móvil. (Generada por Gerardo Fuentes) | 76 |

Lista de cuadros

Resumen

Este proyecto es una primera fase en el desarrollo de un sistema de control alternativo de plataformas móviles. En este caso se contempla el uso de posiciones y/o gestos faciales para poder realizar dicho control. Asimismo, el proyecto busca plantear una propuesta de sistema completo de asistencia para el usuario por medio de la integración de diversas herramientas, como el aprendizaje de máquina, las herramientas gráficas de *Python*, simuladores de fuente abierta, y el paradigma de programación orientada a objetos. En cuanto a los modelos de aprendizaje, se utilizaron redes neuronales convolucionales, que fueron alimentadas con bases de datos de fotografías, donde se realizaron pruebas con y sin filtros computacionales, además de pruebas con y sin marcadores físicos en los usuarios, con el objetivo de optimizar la predicción realizada y verificar el comportamiento al combinar métodos clásicos de reconocimiento facial y métodos modernos (*Machine Learning*). Para obtener dichas fotografías y realizar la predicción de posición en tiempo real, se utilizó tecnología de visión de computadora desarrollada previamente por profesionales, específicamente *OpenCV* y *Mediapipe*. Esta última tecnología permite la captura de posición de puntos de rostros, manos, cuerpo y cuerpo completo, con lo cual, se logró alimentar las redes neuronales de grafos. El siguiente paso fue realizar simulaciones virtuales para verificar que el funcionamiento pueda ser trasladado a una plataforma física. Para poder transmitir los comandos generados luego de la predicción de los modelos de aprendizaje, el estándar de comunicación y la plataforma robótica móvil seleccionada deben adecuarse a los objetivos, en este caso, una comunicación inalámbrica, dado que genera una mayor independencia espacial. Por esta razón, se seleccionó el estándar *Bluetooth* que se encuentra disponible en la placa *ESP32*, la cual se puede comunicar fácilmente con diferentes plataformas robóticas. Teniendo en cuenta la disponibilidad en la Universidad del Valle de Guatemala y los requisitos previamente mencionados, se seleccionó la plataforma robótica Pololu 3pi, la cual puede recibir información de la placa *ESP32*, gracias a la implementación de una placa auxiliar desarrollada en el departamento de Inginería Electrónica, Mecatrónica y Biomédica de la Universidad del Valle de Guatemala. La integración de cada una de las etapas mencionadas anteriormente, logran el objetivo de generar un prototipo de sistema de visión por computadora capaz de detectar gestos/posiciones de la cabeza y así generar movimiento en agentes robóticos, logrando así abrir el camino para futuras aplicaciones de apoyo para usuarios que lo necesiten.

Abstract

This project represents the initial phase in developing an alternative control system for mobile platforms. It explores the use of facial positions and/or gestures to enable such control. Additionally, the project aims to propose a comprehensive user assistance system by integrating various tools, including machine learning, Python's graphical tools, open-source simulators, and the object-oriented programming paradigm. Regarding the learning models, convolutional neural networks were utilized, trained with photo databases. Tests were conducted with and without computational filters, and with and without physical markers on the users, to optimize prediction accuracy and evaluate performance when combining classical facial recognition methods with modern machine learning techniques. For capturing the photographs and making real-time position predictions, computer vision technology previously developed by professionals was used, specifically OpenCV and Mediapipe. The latter allows for capturing the positions of facial points, hands, body, and full body, which fed the graph neural networks. The next step was to perform virtual simulations to ensure the system's functionality could be transferred to a physical platform. To transmit the commands generated after the learning models' predictions, the communication standard and mobile robotic platform selected needed to meet the objectives, in this case, wireless communication for greater spatial independence. Therefore, the Bluetooth standard available on the ESP32 board was selected, as it can easily communicate with various robotic platforms. Considering the resources available at Universidad del Valle de Guatemala and the previously mentioned requirements, the Pololu 3pi robotic platform was chosen, which can receive information from the ESP32 board thanks to an auxiliary board developed in the Department of Electronic, Mechatronic, and Biomedical Engineering at Universidad del Valle de Guatemala. The integration of all these stages achieves the goal of creating a prototype computer vision system capable of detecting head gestures/positions and thus generating movement in robotic agents, paving the way for future applications to assist users in need.

CAPÍTULO 1

Introducción

En primer lugar, para poder realizar este proyecto, fue necesario realizar un análisis de las herramientas capaces de llevarlo acabo y luego la selección de las herramientas no solo capaces sino también disponibles en la región. Con esto en mente, se seleccionó *software* de código abierto, como *Python*, *Webots*, *PyCharm Community Edition*, *TensorFlow*. De igual manera, se escogió hardware al alcance de un estudiante de ingeniería en la Universidad del Valle de Guatemala (UVG) como: *protoboard*, fuentes de alimentación directa, cámara integrada en la *PC*, placa *ESP32*, y agentes robóticos disponibles actualmente en la UVG. El siguiente paso fue determinar que tipo de base de datos se iba a utilizar, dado que se generaron modelos de aprendizaje de máquina. Inicialmente se buscó bases de datos disponibles en la *web*, sin embargo únicamente se encontró uno que cumpliera con las condiciones requeridas: posiciones de cabeza específicas. Por lo que luego de realizar pruebas con los modelos de aprendizaje y no obtener resultados favorables, se procedió a realizar capturas de fotografías con otros alumnos en el campus. Después, se tuvieron que modificar las bases de datos, ya que luego de utilizar fotografías normales, se procedió a obtener capturas de fotografías a usuarios utilizando marcadores físicos en el rostro. Finalmente, se logró utilizar un modelo de aprendizaje que requiere una alta cantidad de fotografías, pero únicamente del usuario a utilizar el sistema.

En cuanto a los modelos de aprendizaje, dado que se busca realizar visión por computadora, se inició realizando pruebas con redes neuronales convolucionales, aplicado a las bases de datos obtenidas de la *web*, y las obtenidas gracias a compañeros en el campus. Con estos modelos, se procedió a realizar ajustes, en cuanto a la cantidad de clases a predecir y las épocas necesarias para entrenar el modelo, buscando siempre un ajuste que devolviera el resultado más óptimo. Al final se definió utilizar únicamente cuatro clases, que corresponden a las posiciones de cabeza: arriba, centro - abajo, derecha e izquierda, dado que más de esta cantidad, daba como resultado una predicción con baja exactitud. Esto sucedió con las bases de datos mencionadas anteriormente, por lo que se buscó cambiar el enfoque de los modelos de aprendizaje. Entonces, se encontró un marco de trabajo desarrollado por *Google* llamado *Mediapipe*, el cual es capaz de detectar diferentes puntos en el rostro de un ser humano, utilizando en conjunto la librería *OpenCV*. Al obtener los datos de dichos puntos, se observó

que estos son capaces de formar un grafo, lo cual provocó las pruebas con redes neuronales de grafos. Al utilizar estos modelos, se realizaron pruebas únicamente con los datos obtenidos del autor (Gerardo Fuentes), y los resultados fueron extremadamente buenos. Dicho resultado llevó a la creación de modelos con más clases, y funcionaron igual de bien, sin embargo, para tener un mejor control se decidió trabajar únicamente con 5 clases, con las siguientes posiciones: arriba, centro, abajo, derecha e izquierda.

Respecto a las simulaciones, al tener los primeros modelos dando resultados prometedores, se procedió a realizar pruebas con *Turtle*, que es una graficadora, la cual se programó para reaccionar dependiendo de las predicciones. Sin embargo esta primera simulación es tan simple como mover un cursor, por lo que se procedió a iniciar con la interfaz gráfica que sería la estructura del sistema de movimiento en general. Mientras los modelos fueron obteniendo resultados aún más prometedores, se realizó una configuración para poder controlar robots simulados de la librería de *Webots*, desde un código de *Python*, usándolo como controlador externo. Una vez se obtuvieron resultados aceptables, se procedió a realizar el cambio de enfoque de redes neuronales convolucionales a redes neuronales de grafos. Al obtener los resultados extremadamente buenos de este modelo, se procedió a realizar cambios en la interfaz gráfica, de tal manera que facilitaran la captura de fotografías del usuario. Luego de optimizada esa sección, se implementaron estos nuevos modelos con la plataforma *Webots*, y nuevamente se obtuvieron resultados favorables, y fue así que se decidió iniciar con las pruebas físicas.

Por último, se seleccionó la comunicación por medio de *Bluetooth*, para transmitir los comandos al agente robótico, dado que se espera que pueda realizar movimientos claros y no mantenerse con movimientos limitados por conexiones alámbricas a la *PC*. En este punto, se realizaron pruebas con diferentes métodos para transmisión con dicho estándar sin necesidad de la configuración de *Windows*. Sin embargo las verificaciones iniciales de estas pruebas, no funcionaron y se optó por emparejar la placa *ESP32*, desde la configuración de *Windows*, y una vez establecida dicha emparejamiento, utilizar los puertos seriales enlazados. En este caso las pruebas fueron favorables, dado que la placa tenía como comandos, la activación de diodos emisores de luz, los cuales reaccionaron justo como se esperaba, dependiendo de la posición de la cabeza. Con esta última verificación exitosa, fue que se realizó la migración de comandos de simulaciones a comandos para el movimiento del robot físico Pololu 3pi+, llegando nuevamente a resultados exitosos.

CAPÍTULO 2

Antecedentes

2.1. Control de gestos para su uso en automóviles

El artículo trata sobre el uso del control de gestos como control para conductores sobre los sistemas integrados en automóviles. Se centra en la detección y reconocimiento de gestos manuales mediante técnicas de recolección y procesamiento de imágenes. En primer lugar se resalta la importancia de tener un contraste entre la mano y el fondo para una detección precisa de gestos. Para lograr esto, se usó un conjunto de LED que emiten luz infrarroja cercana (NIR) con longitud de onda de 950 nm, la cual no es visible para el ojo humano pero si para una cámara CCD modificada. Esta sensor elimina la información de color pero permite la medición de intensidades. [1]

El proceso de recolección de imágenes implica segmentar cada imagen utilizando una única clase o categoría, por medio de umbralización simple. Luego, se rastrean y etiquetan los límites de las regiones conectadas con características iniciales como tamaño, alcance, centroide y momentos de Hu. Estas características ayudan a identificar y distinguir diferentes gestos manuales. Después, en una secuencia de imágenes con duración preestablecida, se emparejan regiones correspondientes (denominadas objetos) entre imágenes consecutivas. Esto se realiza al rastrear los centroides y asumiendo que las regiones pequeñas representan un ruido casi nulo. Por último, se calculan características de movimiento como velocidad y aceleración. Esto permite clasificar de objetos en diferentes clases dinámicas. Para reducir el peso computacional, se aplica un método de preselección basado en puntuación difusa y así eliminar algunos objetos. Entonces se utilizan conjuntos difusos para etiquetar objetos en una clase que indica que no son manos, basándose en el análisis de rangos típicos de características para manos. También incluye una fase de reconocimiento, donde los objetos se comparan con gestos de referencia precargados. En este caso se generan funciones de distribución de probabilidad para cada característica, y se mide la similitud entre el objeto observado y la referencia. Para evitar clasificaciones falsas, se determina un umbral por defecto y, si el objeto muestra poco movimiento, se usa un segundo clasificador basado en correlación de forma. La tasa de procesamiento se configuró en 25 fotogramas por segundo

con una resolución de 192×144 píxeles y una procesador Pentium-II de 333 MHz. [1]

2.2. Reconocimiento de gestos automático para interacciones inteligentes humano-robot

El enfoque principal de este artículo es el reconocimiento de gestos de cuerpo completo para la interacción inteligente entre humanos y robots. Se describe un método basado en aprendizaje que puede realizar simultáneamente la detección y el reconocimiento de gestos importantes. El método implica la estimación de la pose del cuerpo humano en 3D, el entrenamiento de Modelos Ocultos de Markov (HMM, por sus siglas en inglés) para modelar la variabilidad de patrones, y la construcción de un modelo para gestos no realizados. El método propuesto se integra en un sistema robótico llamado T-Rot y logra un alto rendimiento en el reconocimiento. La metodología utilizada fue la siguiente:

2.2.1. Modelo humano en 3D

Se construyó una base de datos jerárquica del cuerpo humano utilizando imágenes de silueta e imágenes de profundidad. El modelo incluye múltiples niveles para representar diferentes posturas del cuerpo humano. [2]

2.2.2. Extracción de características

Basado en información sobre las componentes del cuerpo en 3D se seleccionan trece puntos característicos. También se utilizaron los ángulos desde el eje vertical hasta cada punto característico como una característica. [2]

2.2.3. Fase de entrenamiento

En esta etapa se entrena los HMM para modelar la variabilidad de los patrones. En este entrenamiento se utilizó la base de datos *KU Gesture* y se aplica el algoritmo K-means para dividir los datos de entrenamiento en sub-categorías. [2]

2.2.4. Fase de reconocimiento

En la fase de reconocimiento, se utiliza el algoritmo de Viterbi para encontrar la secuencia de estados más probable para un gesto ingresado al modelo. El gesto se detecta y reconoce en función de los HMMs entrenados. [2]

2.2.5. Resultados experimentales

El método propuesto se evalúa utilizando la base de datos *KU Gesture* y gestos generados. El resultado general de detección es del 94.8 %, y el resultado de reconocimiento de gestos aislados es del 97.4 %. [2]

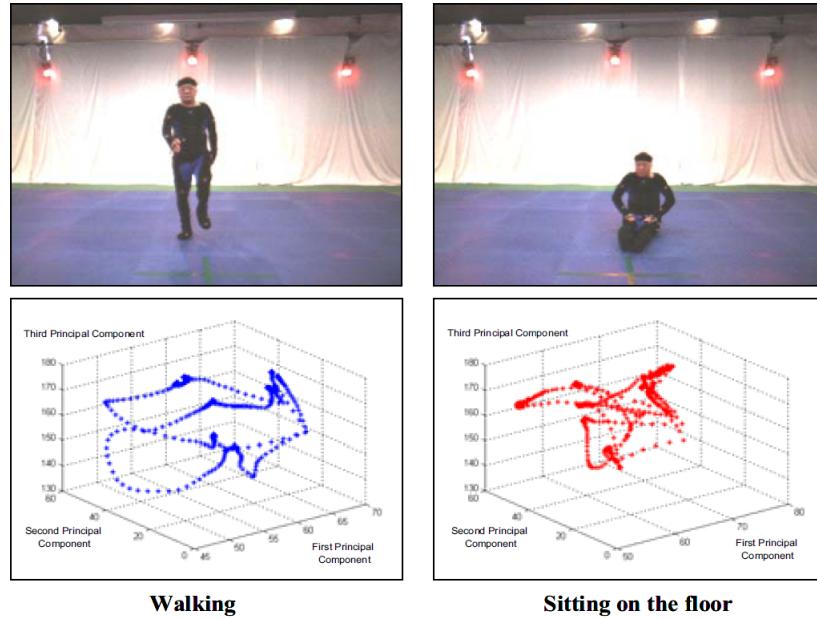


Figura 1: Captura de datos para la trayectoria característica de dos gestos en un espacio cerrado. [2]

2.3. Sistema de reconocimiento de manos y rostro para personas ciegas

Este artículo presenta un sistema de reconocimiento diseñado para ayudar a personas con discapacidad visual. Se han desarrollado sistemas de reconocimiento de gestos con la mano y facial para realizar diversas tareas. El sistema captura imágenes dinámicas de un flujo de video, procesándolas a través de algoritmos respectivos. Para el reconocimiento de gestos con la mano, el sistema primero detecta la región de la mano en imágenes en tiempo real. Esto quiere decir que se realiza conversión del espacio RGB al espacio de color YCbCr. El sistema establece valores umbral superiores e inferiores para la detección de piel, los cuales pueden ser ajustados dinámicamente por el usuario según el entorno. Luego, por medio del algoritmo *Convex Hull* o de envoltura convexa el sistema extrae características de la región de la mano, como las puntas de los dedos y el ángulo entre los dedos. En esta aplicación se reconocen diferentes gestos que representan números del uno al cinco utilizando este sistema. Para el reconocimiento facial, el sistema utiliza clasificadores de cascada Haar y el reconocedor LBPH (Histogramas de Patrones Binarios Locales). La imagen capturada del rostro se convierte a una imagen en escala de grises. Luego, el sistema entrena la base de datos de imágenes utilizando clasificadores y reconocedores. Durante el reconocimiento, la imagen en tiempo real se compara con las imágenes en la base de datos. Si se encuentra

una coincidencia, se muestra el nombre asociado con la imagen. [3]

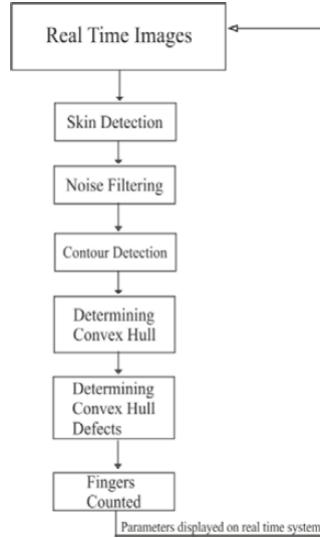


Figura 2: Arquitectura del sistema propuesto para reconocimiento de gestos de manos. [3]

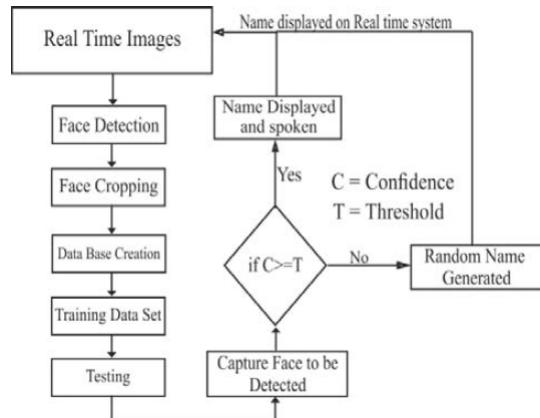


Figura 3: Arquitectura del sistema propuesto para reconocimiento facial. [3]

2.4. Sistema de reconocimiento gestos de cabeza utilizando la cámara de gestos

La Cámara de gestos es una cámara inteligente diseñada para capturar e interpretar gestos de cabeza, así como expresiones faciales. Esta enfocada principalmente en individuos con discapacidades o parálisis. El documento explora diferentes áreas, como reconocimiento de gestos, detección facial, rastreo de rostro, y detección de obstáculos. Este sistema cuenta con unidades de captura de imágenes, reconocimiento de gestos y la de concentración despliegue de datos. La unidad de captura de imágenes utiliza un sensor de imagen a color CMOS, montada en una placa electrónica. Luego de capturar la secuencia de imágenes o

video, la unidad de reconocimiento de gestos analiza e identifica los gestos y expresiones faciales. Para poder realizar esto, utiliza técnicas como modelado de movimiento, análisis de movimiento, reconocimiento de patrones, y *machine learning* para poder interpretar los gestos. Esta unidad también es capaz de detectar el movimiento de la cabeza para luego determinar su orientación. Una vez que los gestos se reconocieron, la información se transfiere a la unidad de concentración y despliegue. Esta unidad puede ser una computadora personal o bien una de control centralizado. La salida de datos de la cámara inteligente, que incluye una descripción de alto nivel del rostro del usuario en diferentes orientaciones, es enviada a la sección de concentración para un futuro procesamiento. [4]

- Los retos mas importantes afrontados en el desarrollo de dicho sistema fueron:
 1. Imágenes fuera del rango de la cámara, esto fue un problema dado que el sistema debe ser capaz de detectar los gestos de movimiento incluso si la cabeza no es completamente visible por el sensor. [4]
 2. La variación de las condiciones de iluminación, el sistema también debe ser capaz de reconocer los gestos de la cabeza con precisión aún si las condiciones de luz cambian, entendiendo que esto puede cambiar el aspecto del rostro del usuario. [4]
 3. La variación de las formas de los rostros, los usuarios jamás tendrán rostros exactamente iguales, tanto por forma, tono de piel, bello facial, gafas, entre otros, dicha variación provoca un mayor reto para reconocer gestos de manera adecuada. [4]
 4. Fondos desordenados, cuando el dispositivo móvil realice una acción de movimiento, el fondo puede parecer desordenado o en movimiento, lo cual se puede mezclar con el movimiento de los gestos, creando así una dificultad para diferenciar el fondo de los gestos en el algoritmo de reconocimiento del sistema. [4]

2.5. Control de una silla de ruedas usando una agrupación de k-medias adaptativas de las poses de la cabeza

En este artículo se presenta la idea de ayudar a las personas con ciertas discapacidades físicas, por medio de una interfaz que utiliza el sensor Kinect desarrollado por Microsoft. El objetivo es permitir a las personas con discapacidades interactuar con los dispositivos electrónicos. Para lograr esto, se utilizó un algoritmo basado en la Regresión de Bosques Aleatorio y uno de agrupación de k-medias para detectar los cambios en el rango de dirección de los movimientos de cabeza. Y la experimentación se realizó en 5 individuos operando la silla de ruedas en condiciones favorables y no favorables. [5]

El sistema propuesto se basa en el método de agrupamiento de k-medias, el cual es un esquema de agrupamiento que utiliza puntos representativos y distancias euclidianas para medir escala de similitud entre vectores y el agrupamiento de puntos representativos. Además los datos obtenidos son posibles gracias al sensor de profundidad utilizado. La arquitectura del sistema propuesto funciona de la siguiente manera:

2.5.1. Calibración

Este proceso consta del establecimiento de la configuración inicial para cada usuario, de tal manera que se pueda realizar un proceso personalizado al usuario. Este proceso requiere que el usuario coloque la posición de la cabeza según solicite la interfaz, para configurar las funciones de detenerse, izquierda, derecha, adelante y atrás. Este proceso utiliza el algoritmo RRF. Luego, los datos recopilados se guardan para luego utilizarse en el proceso de adaptación. [5]

2.5.2. Rectificado

Esta es la etapa final, pero el artículo lo menciona que es importante mencionar en este punto la funcionalidad del proceso. Dado que el sistema se encuentra generando constantemente estimados de los ángulos de pose, es importante realizar un rectificado según la pose y la capacidad de movimiento del usuario. [5]

2.5.3. Adaptación

Se consideró que, dadas las situaciones de fatiga o degeneración en las enfermedades de los usuarios, podría suceder una descalibración del control de movimiento. Para esto se integraron dos métodos al sistema. El primer método consisten en la implementación de sensores de sonar que se encargan de detectar obstáculos. Al detectarlos, se implementa una corrección en el movimiento. El siguiente método consiste en crear agrupaciones adicionales a las de las funciones principales. Dichas agrupaciones, son las combinaciones de los movimientos básicos. Esto es realizado por medio de el algoritmo de agrupamiento de k-medias, y ayuda a no realizar cambios de movimiento abruptos. [5]

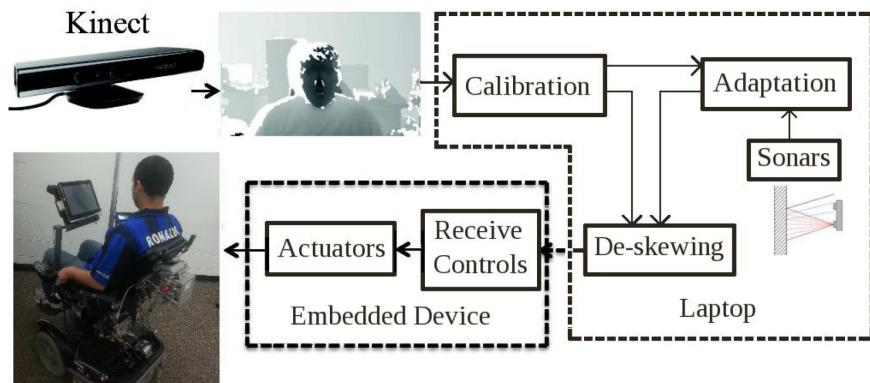


Figura 4: Diagrama del sistema propuesto. [5]

2.6. *Mediapipe*

Mediapipe es una herramienta creada por *Google* que ayuda a construir sistemas inteligentes capaces de entender y procesar información de diferentes tipos, como audio, video y datos en 3D. Ofrece un conjunto integral de herramientas y componentes para construir varios tipos de modelos de aprendizaje automático, especialmente si tienen relación con tareas de procesamiento de multimedia. Una de las ventajas es que puede hacer este procesamiento en tiempo real, lo que es sumamente útil para aplicaciones que requieren alto rendimiento y baja latencia, como aplicaciones de transmisión en vivo y robótica. También, cuenta con compatibilidad multiplataforma, como *Android*, *iOS*, *Linux*, *macOS*, además de *Windows*. Este marco de trabajo tiene una arquitectura modular, por lo que tiene una alta adaptabilidad. *Mediapipe*, es capaz de utilizar modelos preconstruidos para realizar tareas simples, además de permitir integración con modelos de aprendizaje de máquina previamente entrenados con *TensorFlow*. Al ser un marco de trabajo profesional, cuenta con una extensa documentación, y actualización constante.[6]

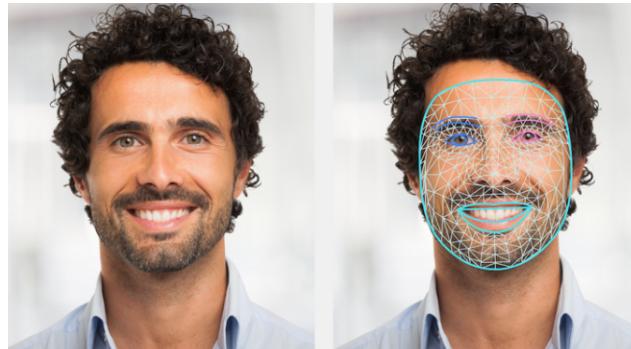


Figura 5: Visualización de el mapeo de punto del rostro utilizando *Mediapipe*. [6]

CAPÍTULO 3

Justificación

Para llegar a ser una sociedad más inclusiva es necesario poder desarrollar tecnología que afronte diferentes situaciones, en este caso las personas con algún tipo de discapacidad. Existe una gran variedad de extremidades inhabilitadas en el espectro de las discapacidades. En muchos casos bastaría con un sistema de reconocimiento de gestos de manos para personas con parálisis de la cintura para abajo. Sin embargo, también hay que tomar en cuenta a las personas con discapacidades en las extremidades superiores, es ahí donde entra el reconocimiento de gestos de la cabeza. El desarrollo de un sistema de visión de computadora capaz de reconocer gestos faciales/de cabeza, que sea capaz de enviar comandos a un robot móvil es el primer paso para desarrollar una tecnología de apoyo. El control de un robot móvil por medio de este algoritmo, dará paso a una futura migración de comandos a plataformas móviles como sillas de ruedas, o bien vehículos eléctricos. Esta alternativa también busca utilizar sensores mucho más comunes en el mercado, como lo son los sensores de imágenes a comparación de sensores de profundidad o bien de señales bioeléctricas. Los sensores de imágenes, además tienen el valor agregado de brindar una mayor libertad de movimiento a los usuarios, ya que no son intrusivos, por lo que el gesto realizado tiende a una mayor naturalidad. Esto sucede dado que al utilizar aquellos que funcionan con EEG o EMG, requieren dispositivos de extensión al cuerpo humano, como son gorras, guantes o electrodos adheridos al cuerpo del sujeto de prueba.

CAPÍTULO 4

Objetivos

Objetivo general

Desarrollar un sistema de visión por computadora para reconocimiento de gestos y orientación de la cabeza para el control de un agente robótico móvil.

Objetivos específicos

- Investigar y seleccionar el hardware y software disponible para la aplicación de visión por computadora.
- Implementar un algoritmo de visión por computadora para reconocimiento de gestos y movimiento de la cabeza.
- Implementar un algoritmo de traducción de los gestos y movimientos de la cabeza a comandos de control para el agente robótico.
- Validar el correcto funcionamiento del sistema de reconocimiento y control por medio de simulaciones computarizadas.
- Validar el correcto funcionamiento del sistema de reconocimiento y control por medio de plataformas robóticas móviles.

CAPÍTULO 5

Alcance

Este control de movimientos por medio de reconocimiento de poses/gestos, busca crear un prototipo listo para pruebas y demostraciones de visión de computadora aplicada a agentes robóticos simples. Dicho esto, el campo físico de trabajo de dicho prototipo se limita a laboratorios de prueba en universidades o espacios cerrados y libres de obstáculos que puedan dañar al prototipo, dado que aún no cuenta con un sistema de seguridad automático. En cuanto al campo de utilización en software, el prototipo se limita a usuarios con software específico instalado, como el lenguaje de programación y sus derivados y el simulador robótico *Webots*, por lo que aún no está listo para usuarios particulares, ni usuarios de otros sistemas operativos diferentes de *Windows*. Siguiendo la línea de las limitaciones en *software*, aún no es un sistema multi-lenguaje, dado que trabaja únicamente con *Python* y sus derivados, y no en otros lenguajes de programación. En cuanto al alcance de tipos de usuarios, el sistema utiliza una tecnología capaz de reconocer la posición de ciertos puntos de la cara respecto del marco de la imagen que captura la cámara. Dicho lo anterior, el sistema es capaz de reconocer rostros humanos sin diferenciar diferentes tonalidades de color, pero sí hay que tener en cuenta que una iluminación delantera pobre en comparación de la trasera, provocará fallos. Además, el sistema trabaja con modelos predictivos únicamente, por lo que no es capaz de generar señales únicamente por medio del modelo, aún requiere una interpretación y traducción a comandos realizada por el usuario. Por último, en cuanto a cantidad de datos, el prototipo es capaz de capturar miles de imágenes en cuestión de minutos, por lo que la limitación está definida por el espacio en disco de la *PC* utilizada.

CAPÍTULO 6

Marco teórico

6.1. *Machine learning*

La inteligencia artificial (IA, por sus siglas en inglés) ha revolucionado la tecnología, y el aprendizaje automático (*Machine learning*) es una de sus ramas más dinámicas y prometedoras. El aprendizaje automático se es al proceso mediante el cual computadoras adquieren la capacidad de aprender y mejorar su rendimiento sin una programación explícita. Esto es posible gracias al análisis de datos e identificación de patrones, que luego permite a las computadoras tomar decisiones, predecir resultados y realizar tareas específicas con un nivel de precisión alto. Se divide en dos categorías principales: supervisado y no supervisado. A nivel empresarial, se utiliza para predecir tendencias de mercado, optimizar procesos de producción y mejorar la experiencia del cliente. En medicina, se usa para el diagnóstico de enfermedades, el descubrimiento de nuevos medicamentos y personalización de tratamientos. En el sector financiero, ayuda a detectar fraudes, gestionar riesgos y automatizar procesos de inversión. Sin embargo, el aprendizaje automático aún presenta muchos desafíos, requiere grandes conjuntos de datos de alta calidad y así como potencia computacional para entrenar modelos efectivos. [7]

6.1.1. Aprendizaje supervisado

El aprendizaje supervisado implica entrenar un modelo utilizando datos etiquetados para hacer predicciones o tomar decisiones con precisión basadas en datos pasados. El conjunto de datos etiquetados sirve como guía para que la computadora comprenda las relaciones entre los datos de entrada y las etiquetas de salida correspondientes. A través de un entrenamiento iterativo (por ejemplo redes neuronales), el modelo aprende a hacer un mapeo de los datos de entrada hacia las etiquetas de salida correctas, lo que le permite predecir con precisión cuando se presenta nueva información. El modelo identifica patrones y asociaciones dentro de los datos basados en características como forma, tamaño u otras características para luego, cuando se introduce nueva información no etiquetada, el modelo aprovecha su

conocimiento aprendido para predecir la salida correcta en función de la similitud con los datos etiquetados que sirvieron de entrenamiento. El aprendizaje supervisado puede utilizarse para dos funciones principales: clasificación y regresión. La clasificación se utiliza cuando la salida tiene la característica de pertenecer o no a una categoría. Por otro lado, la regresión se emplea cuando la salida es un valor continuo, por ejemplo, para predecir el salario en función de la experiencia laboral, edad, estudios, etc. [8]

6.1.2. Aprendizaje no supervisado

En el aprendizaje no supervisado los algoritmos identifican de forma autónoma patrones y estructuras dentro de datos no etiquetados. Las técnicas de aprendizaje no supervisado abarcan diferentes métodos, como el análisis de *clusters*, la detección de anomalías y las redes neuronales, con el propósito de descubrir patrones ocultos y relaciones dentro de conjuntos de datos. El análisis de *clusters*, agrupa puntos de datos con características similares para detectar estructuras y relaciones entre los datos analizados. Esta técnica ayuda a comprender los agrupamientos naturales presentes en los datos, permitiendo obtener información sobre patrones que pueden no ser evidentes a simple vista. La detección de anomalías, se centra en identificar valores atípicos o irregulares en los datos que se desvían significativamente de la norma. Las redes neuronales son importantes especialmente en tareas generativas y clasificativas. Las generativas implican modelar la distribución de los datos para que la red genere nuevas muestras, mientras que las clasificativas se centran en distinguir entre diferentes clases o categorías dentro de los datos. Las redes neuronales en el aprendizaje no supervisado aprovechan técnicas como *autoencoders*, máquinas de Boltzmann restringidas, entre otras para aprender representaciones de datos sin etiquetas. [9]

6.2. Redes Neuronales

Las redes neuronales artificiales son un tipo de modelo de *Machine Learning* que busca mimetizar la estructura y función de una red neuronal biológica. Se componen de nodos interconectados o también llamados neuronas artificiales, las cuales se encargan de procesar y transmitir información a lo largo de la red. Cada nodo recibe datos de ingreso a los cuales se les agrega un peso o indicador de importancia, y así la salida de esta neurona se transmite a lo largo de otras, y este proceso es llamado propagación. Este peso se ajusta según el error calculado entre la predicción realizada por la red neuronal y el valor real, el cual se provee con el nombre de *Datos de entrenamiento*. Todo este proceso, es conocido como el aprendizaje, y es el método que se utiliza para mejorar el reconocimiento de patrones y predicción de resultados. La inspiración biológica proviene de la forma en que las neuronas se conectan entre sí y envían impulsos eléctricos dependiendo del estímulo, permitiendo así el aprendizaje. De manera similar las redes neuronales artificiales aprenden ajustan los pesos mencionados anteriormente. Por esto, las redes neuronales son específicamente útiles cuando se trabaja con patrones complejos y series de datos de gran tamaño. Normalmente se utilizan para el reconocimiento de imágenes, de voz, y procesamiento de idiomas. Otro punto importante, es la capacidad de combinar modelos para crear otros más complejos, para realizar funciones igual de complejas. Aún así, las redes neuronales tienen sus propias limitaciones, en primer lugar, necesitan de grandes series de datos y poder computacional

para realizar un aprendizaje efectivo, y aún así pueden caer en el fenómeno del sobreajuste. [10]

6.2.1. Estructura básica de las redes neuronales

El perceptrón

Es la red neuronal más simple de todas, contiene únicamente una capa de entrada y un nodo de salida. Este realiza cálculos para la detección de patrones en los datos ingresados en la capa de entrada, normalmente se utiliza el aprendizaje supervisado para la clasificación. Matemáticamente se trata de una función que busca predecir correctamente la clase correspondiente para cada valor de entrada ingresado. El objetivo del perceptrón es poder encontrar el vector de pesos (importancia), que sea capaz de hacer la diferenciación de clases. A continuación se puede observar una representación gráfica y matemática de la estructura mencionada:

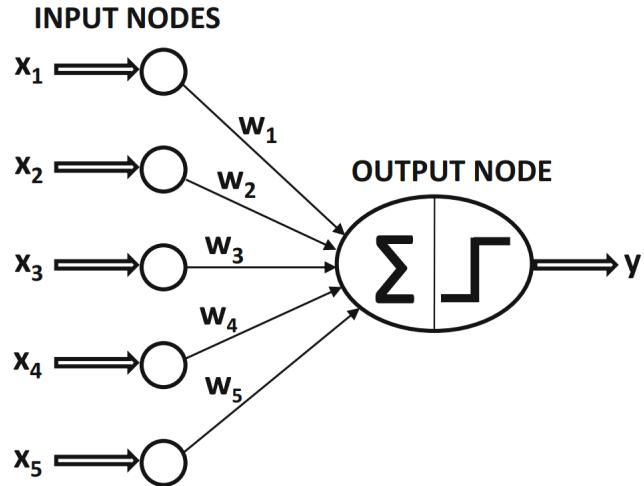


Figura 6: Estructura gráfica de del funcionamiento del perceptrón [10]

$$\hat{y} = \text{sign} \left\{ \bar{W} * \bar{X} \right\} = \text{sign} \left\{ \sum_{j=1}^d w_j x_j \right\}$$

Tanto en la figura como en la ecuación, \bar{W} representa el vector de pesos (importancia) mencionados anteriormente, y cada valor del vector representa que tan importante es dicha característica dentro de la clasificación de una clase en específico. Ahora bien la \bar{X} representa el vector de características, o también conocido como los datos de entrada. La función *sign*, es la que se encarga de categorizar dependencia de cada característica para par clase dada, esto lo hace por medio de una binarización con valores -1 o bien 1 . Sin embargo, aunque en teoría la función presentada funciona bien, esto solo ocurre cuando la información o datos de ingreso son linealmente separables. Es por esto que se desarrollaron diferentes funciones de activación y pérdida, que puedan manejar la clasificación de datos no lineales, lo cual es sumamente importante cuando se trabaja con redes de multicapa. [10]

A continuación se muestra una serie de gráficas de funciones de activación utilizadas frecuentemente:

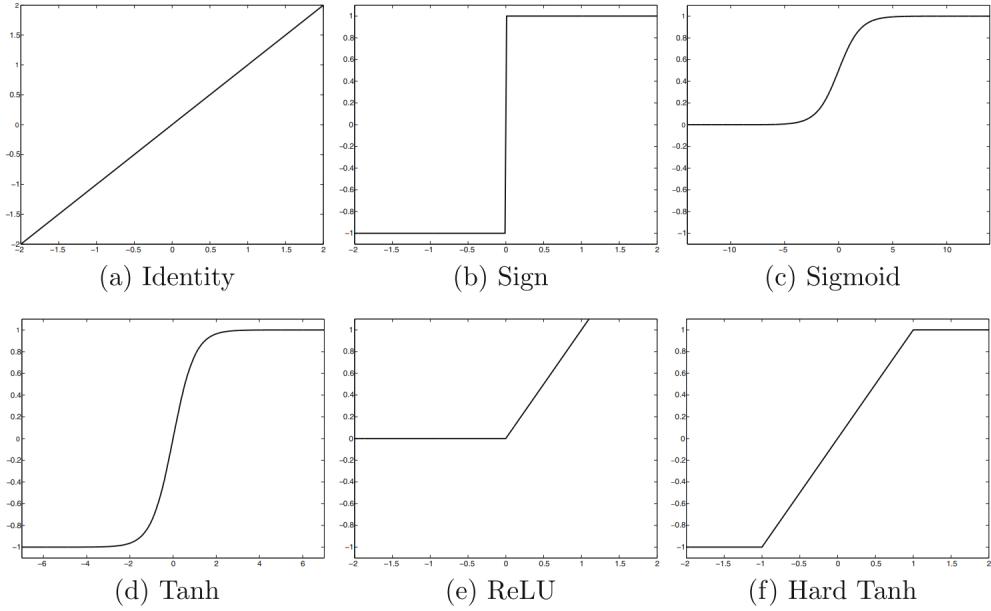


Figura 7: Gráficas varias de funciones de activación [10]

Redes multicapa

A diferencia del perceptrón, estas redes contienen capas ocultas, donde los cálculos computacionales no son visibles para el usuario. Normalmente, estas redes son conocidas como *feed-forward*, lo cual significa que el flujo de información o datos van desde la entrada a la salida por medio de capas sucesivas. En este caso, la arquitectura requiere especificar el número de capas, la cantidad y tipo de nodos en cada capa, por último las funciones de activación por capa y las funciones de pérdida. Nuevamente las conexiones, representan las matrices de pesos, que son determinadas por la salida de cada capa, por medio de las funciones de activación. Asimismo, es importante establecer que cada modelo de red neuronal debe ajustarse a las necesidades del proyecto, tal es el caso de modelos con capas no consecutivas o bien con múltiples salidas. [10]

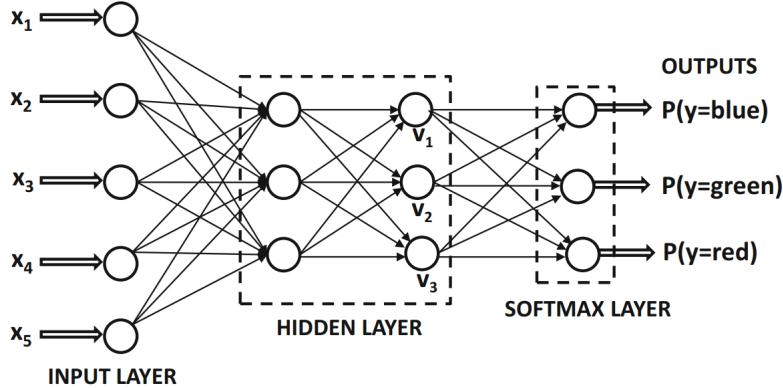


Figura 8: Representación de una red multicapa para clasificación de colores[10]

6.2.2. Redes neuronales convolucionales

Las redes neuronales convolucionales (CNN por sus siglas en inglés) representan uno de los mayores avances en el campo de la inteligencia artificial, especialmente en la visión por computadora. Estas se construyen de tal manera que buscan reflejar el sistema visual humano, permitiendo a las máquinas interpretar y comprender datos visuales con alta precisión. Las CNN están estructuradas para extraer y aprender representaciones jerárquicas de características a partir de los datos de píxeles de fotografías, lo que hace posible tareas como la clasificación de imágenes, la detección de objetos y la segmentación. Las CNN tienen varias capas en su estructura, cada una con diferentes funciones en el proceso de extracción y clasificación de características. La capa convolucional, que es la principal, trabaja similar a un conjunto de filtros que convolucionan a través de la imagen de entrada, detectando patrones y características en diferentes ubicaciones de dicha imagen. Es así como la red comienza a diferenciar características como bordes y texturas, generando gradualmente representaciones más complejas en las capas sucesivas. Luego de las capas de convolución, se colocan las capas de agrupación, que sirven para disminuir las dimensiones espaciales de las características, logrando así un control efectivo respecto al número de parámetros para prevenir el sobreajuste. En estas capas se selecciona el valor máximo o promedio dentro de regiones localizadas, de esta manera retienen las características más importantes mientras descartan información redundante, logrando invariancia espacial y en traslación. Conforme avanza la información en las capas se realiza la clasificación y funciones de activación como *ReLU* introducen no linealidad en la red, lo cual permite modelar relaciones complejas entre características y etiquetas. Para entrenar una CNN es necesario ingresar conjuntos de datos etiquetados y ajustar sus parámetros repetidamente para minimizar el error en las predicciones. Este proceso, también utiliza técnicas como la retropropagación y el descenso de gradiente, para facilitar el reconocimiento. [11]

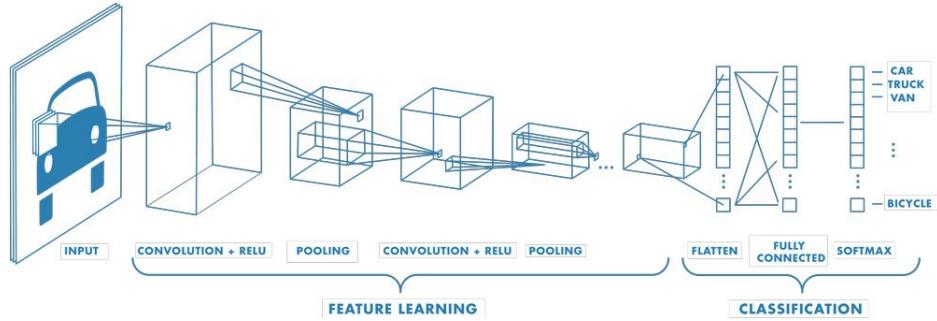


Figura 9: Representación del proceso de las redes convolucionales [12]

6.2.3. Redes neuronales de grafos

Las redes neuronales de grafos (GNN, por sus siglas en inglés) son un tipo de modelo de aprendizaje profundo que se ha convertido en una herramienta poderosa para analizar y procesar cualquier tipo de datos que sea capaz de formar grafos. Se diferencian de las redes neuronales tradicionales dado que operan con datos en formato de vector o matriz, dado que trabajan con estructuras de datos más complejas, como redes sociales, moléculas químicas o sistemas de transporte. El principio fundamental de las GNN radica en la capacidad de capturar las relaciones y dependencias existentes dentro de la información ingresada en forma de grafo. Por medio de la aplicación de operaciones de convolución y agregación respecto de los nodos y aristas, pueden aprender representaciones vectoriales de los elementos del grafo que reflejen sus propiedades estructurales. Esto permite que se puedan realizar tareas como clasificación de nodos, predicción de enlaces y generación de grafos de manera más efectiva que los enfoques tradicionales. El proceso de entrenamiento implica la propagación de información a través de la estructura del grafo, de manera similar a como se propaga la activación en una red neuronal tradicional. Inicialmente, se tiene que establecer una matriz de adyacencia en la que se establece la relación entre cada nodo por medio de una binarización realizada en cada posición de la matriz, acorde a la posición del nodo. Luego, en cada capa, los nodos actualizan sus representaciones vectoriales en función de las representaciones de sus nodos vecinos y de las características de las aristas que los conectan. Algunas de las aplicaciones más destacadas de las GNN incluyen el reconocimiento de patrones en redes sociales, la predicción de interacciones en sistemas biológicos, la clasificación de moléculas químicas y el modelado de sistemas de transporte. Además, las GNN han demostrado ser especialmente útiles en tareas que involucran datos con estructura de relaciones, donde las conexiones y dependencias entre los elementos son fundamentales para comprender el sistema. [13]

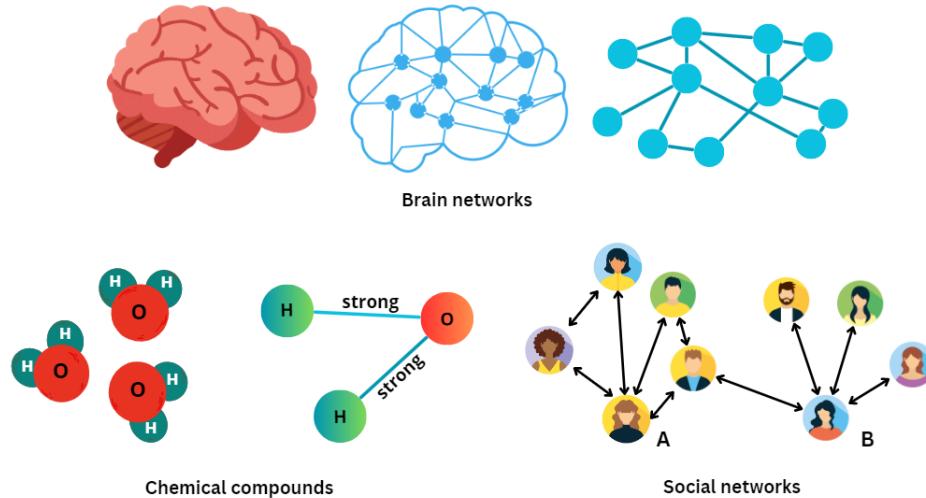


Figura 10: Ejemplos de estructuras capaces de formar grafos[14]

6.3. Visión por computadora

Como humanos percibimos el mundo tridimensional con bastante facilidad, esto es ejemplificado al momento de hacer la diferenciación entre los detalles de macetas o bien contar personas o interpretar sus emociones en un retrato grupal. Replicar este proceso es una tarea que tanto los psicólogos e investigadores de visión por computadora han intentado por años. Mientras que las ilusiones ópticas nos ofrecen algunos indicios, el entendimiento por completo aún es incierto. En el mundo de la visión por computadora, se han hecho avances en técnicas matemáticas para reconstruir formas e imágenes tridimensionales, el seguimiento de objetos, e incluso el reconocimiento de personas en fotografías. Sin embargo, alcanzar una interpretación de imágenes comparable a la de niños pequeños aún es un reto. La visión posee dificultades dado que es un problema de ingeniería inversa, el cual requiere modelos basados en física y estadística para navegar en la ambigüedad. Actualmente, la visión por computadora tienen sus cimientos en dichos modelos de física y gráficas de computadora, abordando problemas sobre cómo se mueven los objetos, el reflejo de la luz e interacción con el ambiente. Hoy, la visión de computadora es subestimada por personas externas debido a una malentendido con respecto al funcionamiento, esta es una creencia refutada, esta afirmaba que la percepción era más simple que el pensamiento cognitivo. En la actualidad, existen muchas aplicaciones de visión por computadora, las cuales tienen diferentes retos respecto de su complejidad en matemáticas, o de la naturaleza misma del problema. En visión de computadora, es sumamente importante aplicar métodos o técnicas que se acoplen al problema o reto en vez de producir métodos genéricos. De esta manera se genera un pensamiento ingenieril para la resolución de problemas, dado que muchas veces se requiere modelos de la física del caso y así como la estadística para modelar escenarios y el ruido que se pueda obtener un escenario más fiel a la realidad. [15]

El campo de la visión por computadora es una ramificación de la inteligencia artificial, la cual tiene el propósito de utilizar computadoras y sistemas para poder obtener información importante desde imágenes, videos u otros datos visuales, para luego poder realizar acciones.

Este campo es un sistema que se forma con componentes de *hardware*, los cuales pueden ser cámaras, iluminación, lentes, sensores de imágenes, etc. Además, también deben poseer una sección de *software*, que se refiere a los algoritmos para procesar y analizar imágenes, videos, etc. También, es necesario tomar en cuenta la comunicación entre el *hardware* y *software*, dado que esto es lo que permite el funcionamiento del sistema, para esto existen diferentes protocolos. [16]

6.3.1. Procedimientos

Clasificación de imágenes

Es el proceso de categorizar o etiquetar imágenes en diferentes clases o categorías predefinidas. Esto incluye el entrenamiento de un modelo de *Machine Learning* para reconocer y diferenciar entre los diferentes objetos, escenarios, o patrones de imágenes. El objetivo es desarrollar un modelo que pueda clasificar de manera eficaz imágenes nuevas o no antes vistas, basado en los patrones y características que se han aprendido previamente. La clasificación de imágenes tiene muchos campos, como detección de objetos, detección de rostros, imágenes médicas, y vehículos autónomos. El proceso para realizar la clasificación de imágenes sigue de la siguiente manera:

Recolección de datos: En esta etapa se debe recolectar un grupo grande de imágenes las cuales estén previamente etiquetadas en clases o categorías específicas.[17]

Procesamiento de datos: Esta parte se encarga de limpiar y realizar un proceso previo a las imágenes de tal manera que aseguren que tienen el formato adecuado para ingresar al modelo. Esta etapa incluye, reajuste de tamaño o proceso de normalizado. [17]

Extracción de características: En este caso se extraen características únicas de las imágenes que ayuden a distinguir entre clases, esto se puede hacer con diferentes técnicas como redes neuronales convolucionales o con otro métodos menos automáticos. [17]

Entrenamiento del modelo: Para poder evaluar el modelo, se necesita un grupo de datos separado, para verificar el rendimiento. Las métricas de evaluación normalmente son: exactitud, precisión, *recall*, y la calificación F1. [17]

Optimización del modelo: Ajustar el modelo es posible utilizando los hiperparámetros, como la optimización del proceso de entrenamiento, o usando técnicas como la regularización.[17]

Predictión: Ya que el modelo está entrenado y optimizado, se puede usar para clasificar nuevos datos, o datos no antes vistos, y este devuelve la categoría o clase en la que fue clasificado el dato o imagen. [17]

Detección de objetos

La detección de objetos es una técnica de visión por computadora para localizar instancias de objetos en imágenes o videos. Los algoritmos de detección de objetos suelen aprovechar el aprendizaje automático o el aprendizaje profundo para producir resultados significativos. La detección de objetos es una tecnología clave detrás de los sistemas avanzados de asistencia al conductor (ADAS) que permiten a los automóviles detectar carriles de conducción o realizar la detección de peatones. La detección de objetos también es útil en aplicaciones como sistemas de vigilancia de video o sistemas de recuperación de imágenes. [18]

Estimación de pose

Es el proceso de determinar la posición y orientación de un objeto en un sistema de coordenadas. Se usa comúnmente en visión de computadora y aplicaciones robóticas. En el contexto de estimación de pose de cámaras, el objetivo es determinar la posición y orientación de una cámara relativa al escenario tridimensional. Esto se realiza normalmente, por medio del emparejamiento de los puntos en imágenes 2D con sus puntos correspondientes en el espacio 3D. Este proceso incluye resolver el problema de la Perspectiva en punto o *Perspective-n-Point*, el cual implica encontrar la pose de una cámara dado un grupo de puntos correspondientes entre el plano 2D y el espacio 3D. Para realizar la estimación de pose existen muchos métodos, incluyendo iterativos y no iterativos. Los métodos iterativos realizan una refinación del estado inicial por medio de un proceso de estimación. Por otro lado, los métodos no iterativos calculan la pose una única vez, sin iteraciones. Un método popular no iterativo es el algoritmo EPnP (*Efficiente Perspective-n-Point*), que está basado en la resolución de un grupo de ecuaciones lineales. Esta utiliza una suma ponderada de puntos de control virtuales para representar puntos en tres dimensiones y reducir el problema a la estimación de las coordenadas de dichos puntos. Este método es computacionalmente eficiente y devuelve resultados precisos. [19]

Rastreo de objetos

En visión de computadora el rastreo de datos involucra la aplicación de algoritmos para detectar y seguir el movimiento de objetos específicos dentro de un video o a lo largo de cuadros. Este proceso es esencial para varias aplicaciones, desde el rastreo de robots en bodegas hasta el rastreo en sistemas de drones. El rastreo tiene sus pilares en la detección de objetos, sin embargo el objeto puede tener distintas apariencias dependiendo de los escenarios y los ángulos. El algoritmo de rastreo predice la posición de un objeto en cuadros de imagen consecutivos, al mismo tiempo que identifica y sigue más de un objeto en una imagen o video. El proceso de rastreo de objetos inicia con la definición del objeto de interés, esto se hace encerrando al objeto en un recuadro, en el primer cuadro del video. El algoritmo de rastreo, luego estima o predice la posición en los cuadros restantes mientras vuelve a encerrar al objeto en ese recuadro de manera simultánea. El algoritmo para rastreo de objetos puede clasificarse basado en el número de objetos que rastrean, incluyendo el rastreo de un único objeto, el cual implica rastrear un objetivo a la vez, y el rastreo múltiple de

objetos. Los algoritmos de rastreo tienen muchos retos como ambientes complejos, y la aplicación de diferentes técnicas, como el aprendizaje profundo. El rastro de objetos es una tarea importante en la visión de computadora por sus aplicaciones en, monitoreo de tráfico, robótica, imágenes médicas, etc. [20] [21] [22] [23]

Reconocimiento de gestos

La interacción entre computadoras y humanos existe de diferentes maneras, pero la interacción por medio de gestos tiene un peso importante dado que, le permite a los humanos una mayor naturalidad al momento de comunicarse. Hoy, existen diferentes métodos para reconocimiento de gestos, los cuales requieren dispositivos que sirven como extensión al cuerpo humano. Sin embargo, estos limitan la naturalidad con que un sujeto de prueba puede interactuar con su ambiente. La alternativa que se propone, desde hace más de veinte años, es la visión de computadora, dado que para capturar los gestos de la persona únicamente se requiere una cámara . Actualmente, el reconocimiento de gestos por visión de computadora está basado en su mayor parte en el *Hidden Markov Model (HMM)*, los algoritmos de redes neuronales y el algoritmo de redondeo del tiempo dinámico. Los pasos para realizar dicho reconocimiento, son: recopilación de imágenes, segmentación de manos, reconocimiento de gestos y clasificación. [24]

La recolección de imágenes se divide en dos partes fundamentales, RGB y de profundidad. Comúnmente, se utiliza únicamente cámaras para detectar el RGB, sin embargo, para la profundidad es necesario utilizar otro tipo de sensores, los cuales puedan devolver información de profundidad. Además, para poder analizar correctamente los gestos es necesario también dividirlos entre estáticos y dinámicos. Dicha división se refiere a la distinción entre fotos o videos respectivamente. Lastimosamente, el proceso de recolección de imágenes se ve afectado por problemas de dirección e intensidad de luz, por lo que los algoritmos realizados deben apuntar a ser invariantes respecto iluminación. [24]

6.3.2. Hardware

Cámaras o sensores de imágenes

Los sensores de las cámaras digitales son dispositivos electrónicos que contienen millones de píxeles fotosensibles, estos registran la cantidad de luz que reciben. Al presionar el botón para tomar la fotografía, se descubren los píxeles para recoger los fotones y almacenarlos como una carga eléctrica. Cuando termina la exposición a la luz, la cámara cierra cada uno de estos píxeles y luego revisa cuántos fotones cayeron, midiendo así, la intensidad de la señal eléctrica. Estos datos se guardan como valores digitales, y su precisión es determinada por la profundidad de bits. Luego, los datos se procesan y convierten en una imagen que se guarda en una tarjeta de memoria o en la cámara por sí misma. En la actualidad, estos sensores se utilizan en cámaras digitales, módulos de cámara, teléfonos con cámara, equipos de imágenes médicas, dispositivos de visión nocturna, etc. En productos de consumo diario normalmente utilizan sensores *Semiconductores complementarios de óxido metálico* o CMOS, por sus siglas en inglés, los cuales son en su mayoría más baratos y su consumo de energía es bajo en dispositivos con batería. Por otro lado, los sensores *Dispositivos de carga acoplada* o

CCD, por sus siglas en inglés, se utilizan en cámaras de video de alta calidad. Ambos tipos de sensores cumplen la misma tarea de capturar luz y convertirla en señales eléctricas. Los sensores de imagen también pueden tener diferentes tamaños y resoluciones, lo cual afecta la calidad de la imagen. Los sensores más grandes tienen una mayor capacidad para capturar luz y, por ende, pueden producir imágenes de mayor calidad y con menor ruido. Además, los sensores de imagen también pueden tener diferentes formatos, como el *Full-frame*, el formato APS-C y el formato *Micro Four Thirds*, que afectan el ángulo de visión y la profundidad de campo. [25]

Sensor LiDAR

LiDAR, o *Light Detection and Ranging*, es una tecnología de teledetección, y funciona al emitir pulsaciones láser y luego midiendo el tiempo que tarda la luz en regresar después de golpear un objeto o la superficie terrestre. Al medir el tiempo de los pulsos con precisión, los sistemas LiDAR pueden calcular la distancia al objeto o superficie. Este proceso se repite varias veces para crear un mapa tridimensional detallado. Los datos resultantes suelen representarse como un sistema de coordenadas tridimensional. En los vehículos autónomos, LiDAR es un componente importante que les permite percibir y navegar su entorno con alta precisión y exactitud. En robótica, se utiliza para la localización y mapeo simultáneos, lo que permite a los robots crear mapas de su entorno y navegar dentro de él. También, se usa para crear mapas altamente detallados del terreno, vegetación y cobertura terrestre, lo cual ayuda a la gestión ambiental y de agricultura. En la planificación urbana, los datos se utilizan para crear modelos tridimensionales detallados de ciudades, para evaluar infraestructuras. Las ventajas de este sistema incluyen: datos tridimensionales altamente precisos y detallados, que lo hace valioso para diversas aplicaciones que requieren información espacial. También, puede utilizarse en diversos entornos y para muchas aplicaciones, como la gestión ambiental y desarrollo de infraestructuras mencionadas anteriormente. Además tiene la capacidad de capturar rápidamente grandes cantidades de datos, lo que permite un eficiente mapeo y topografía de áreas extensas. Por otro lado, sus mayores desventajas radican en el alto costo y en la baja compatibilidad de los datos recopilados. [26]

Sensor RADAR

Estos dispositivos convierten las señales de eco de microondas en señales eléctricas, utilizando tecnología inalámbrica para detectar el movimiento por medio de la posición, forma, tipo de movimiento y trayectoria de un objeto. A diferencia de otros sensores, no se ven afectados por el espectro de luz, y tienen la capacidad de detección aunque existan obstáculos de por medio. Una de sus principales ventajas es su capacidad para detectar el movimiento al calcular la velocidad y dirección de un objeto a través del efecto Doppler, así como observar el movimiento del objetivo desde diferentes perspectivas usando sensores multicanal. Los sensores de radar se utilizan cada vez más en dispositivos portátiles, edificios inteligentes y vehículos autónomos, y su importancia histórica es evidente en aplicaciones como la predicción del clima y el seguimiento de la temperatura. [27]

Estos sensores funcionan detectando la radiación electromagnética transmitida y reflejada. Envían pulsos electromagnéticos cronometrados y analizan la diferencia entre las señales

enviadas y recibidas para detectar el movimiento, la presencia, la distancia, la velocidad y la localización de objetos. Estos sensores se utilizan en una gran variedad de campos donde se requiere una acción automática al detectar un objeto en movimiento, como controlar luces, sistemas de calefacción y refrigeración, dispositivos de encendido/apagado y abrir o cerrar puertas. Además, uno de los objetivos clave al utilizar sensores de radar es reducir el consumo de energía. [28]

Cámaras *Time-of-Flight (TOF)*

Una cámara con esta tecnología opera al iluminar el espacio o escena con una fuente de luz modulada, normalmente con un láser de estado sólido o un LED operando cerca del rango infrarrojo, por ende la luz es invisible al ojo humano. Luego, la cámara percibe la luz reflejada y mide el desfase entre la iluminación y la reflexión. Este desfase es usado para determinar la distancia entre objetos dentro del espacio. Las cámaras del TOF utilizan arreglos de píxeles de CMOS de bajo costo y una fuente de luz activa y modulada. La luz que entra a la cámara tiene tanto un componente ambiental como un componente de reflejado. La información de distancia está embebida en el componente reflejado de la luz. Sin embargo, un componente de ambiente robusto puede reducir la razón de señal-ruido de la cámara. Para poder detectar el desfase, la fuente de luz es modulada de manera continua por medio de una fuente de onda continua, como lo es un senoide o bien una señal cuadrada. La modulación por señal cuadrada es más común, dado que se puede realizar fácilmente utilizando circuitos digitales. También, puede ser generada al integrar los fotoelectrones desde la componente de luz reflejada o usando un contador rápido que sea activado por la primera detección de reflexión. Las cámaras con tecnología TOF también incluyen un controlador (TFC), que es una máquina de estados que sincroniza la operación del sensor, la interfaz analógica, y la iluminación. El controlador escanea los píxeles, calcula la profundidad para cada uno, y realiza varias tareas de procesado como eliminación de aliasing, de ruido, afinación de frecuencias, y compensación de temperatura. También maneja la alta tasa de entrada y salida, serialización y deserialización. [29]

Kinect

El sensor Kinect, es un sensor de profundidad que opera con base en el principio de luz estructurada y aprendizaje automático. Consiste en una cámara infrarroja y un emisor de luz, que proyecta un patrón conocido de luz infrarroja sobre el espacio. El sensor captura la deformación de este patrón, que luego se utiliza para calcular el mapa de profundidad de los objetos en el espacio. Este proceso de cálculo del mapa de profundidad implica analizar el patrón punteado de la luz láser infrarroja. Esta técnica de análisis de un patrón conocido se llama luz estructurada. El principio general es proyectar un patrón conocido sobre el espacio e inferir la profundidad a partir de la deformación de ese patrón. El mapa de profundidad se construye analizando la deformación del patrón de punteado de la luz láser infrarroja. El cálculo de la profundidad se realiza mediante el hardware *PrimeSense* incorporado en Kinect, que utiliza triangulación para calcular el mapa de profundidad. La segunda etapa del proceso implica inferir la posición del cuerpo utilizando el aprendizaje automático. Esta etapa perfecciona el mapa de profundidad obtenido del análisis de luz estructurada, teniendo en cuenta la perspectiva y otros factores para mejorar la precisión de la información de

profundidad. El sensor Kinect ha sido evaluado para diversas aplicaciones de visión por computadora, incluyendo resolución y precisión 3D, ruido estructural, configuraciones de múltiples cámaras y respuesta transitoria del sensor. [30]

6.3.3. Software

OpenCV

OpenCV (*Open Source Computer Vision*) es una biblioteca de funciones de programación que se utiliza principalmente para el procesamiento de imágenes. Contiene una API estándar (Interfaz de Programación de Aplicaciones) para aplicaciones de visión por computadora. Al inicio, OpenCV fue desarrollado como un proyecto de investigación de Intel, pero ahora está disponible de manera gratuita bajo la licencia de Distribución de Software de Berkeley. Está escrito principalmente en C++, pero también se puede programar en Python, Java y MATLAB. Es compatible con varias plataformas, incluyendo Windows, Android, iOS, Linux y más. Además, está en constante evolución, con investigación y desarrollo continuos. El propósito principal de OpenCV es ayudar a las computadoras a entender el contenido de las imágenes, por lo que contiene una gran variedad de herramientas y algoritmos que pueden utilizarse para resolver diversos problemas de visión por computadora. Algunas de las herramientas son:

Filtrado de imágenes: Contiene funciones para modificar o mejorar imágenes con técnicas como filtrado de imágenes lineales y no lineales. Esto se puede utilizar para eliminar ruido, desenfocar o enfocar imágenes entre otras. [31]

Transformación de imágenes: También tiene métodos para generar nuevas imágenes a partir de varias fuentes, destacando características o propiedades específicas. Esto incluye técnicas como la Transformada de Hough para encontrar líneas en una imagen, la Transformada de Radón para reconstruir imágenes a partir de datos de proyección y otras transformadas para compresión de imágenes y videos. [31]

Detección de características: Existen herramientas para encontrar características específicas en una imagen, como líneas, bordes o ángulos. Estas características se pueden utilizar como base para otros algoritmos de visión por computadora y son importantes para tareas como el reconocimiento y rastreo de objetos.[31]

Rastreo de objetos: También incluye algoritmos para localizar y rastrear objetos en una secuencia de imágenes. Esto es útil en aplicaciones como vigilancia, interacción humano-computadora e imágenes médicas.[31]

Detección y reconocimiento de rostros: OpenCV posee modelos y algoritmos pre-entrenados para detectar y reconocer rostros en imágenes y videos. Esto se puede usar en

aplicaciones como autenticación facial, donde se compara una imagen capturada con una base de datos de rostros conocidos.[31]

TensorFlow

Este es un sistema para aprendizaje automático a gran escala, el cual fue desarrollado Google. Está diseñado para soportar el entrenamiento e inferencia de modelos de aprendizaje automático en diferentes plataformas, desde clústeres hasta dispositivos móviles. TensorFlow utiliza una representación conceptual de flujo de datos para representar tanto la computación en un algoritmo como el estado en el que se encuentra operando el algoritmo. También, ofrece un modelo de programación flexible y general para la investigación en aprendizaje automático. Este sistema ha sido ampliamente utilizado en producción para Google y también fue lanzado como un proyecto de código abierto. TensorFlow puede utilizarse en diversas aplicaciones, incluyendo la visión por computadora al proporcionar un marco de trabajo robusto para construir y entrenar modelos de aprendizaje profundo, los cuales se utilizan ampliamente en visión por computadora. Además, se puede implementar y experimentar con algoritmos de visión por computadora de última generación, entrenar modelos con conjuntos de datos extensos y utilizarlos para clasificación de imágenes, detección de objetos, segmentación de imágenes, etc. [32]

CUDA

El software NVIDIA CUDA es un modelo de programación y plataforma que permite a los desarrolladores aprovechar las tarjetas gráficas de NVIDIA para diversas aplicaciones. Se basa en una arquitectura de cómputo paralelo y proporciona un modelo de programación que demuestra eficientemente dicho paralelismo de datos. El software permite escribir código que puede ejecutarse en los procesadores paralelos de las GPU de NVIDIA, esto permite procesos de alto rendimiento y la aceleración de diversas aplicaciones. Para visión de computadora el software NVIDIA CUDA es considerado relevante ya que permite aprovechar el paralelismo al analizar datos. CUDA permite el trabajar de manera eficiente grandes cantidades de datos visuales, como procesamiento de imágenes y videos, detección y rastreo de objetos. Al utilizar CUDA, se puede acelerar el rendimiento de algoritmos de visión por computadora, por medio del paralelismo. Una desventaja para esta aplicación, es que requiere una tarjeta gráfica de NVIDIA, que no siempre se encuentra disponible. [33]

MATLAB

Es una plataforma de computación numérica y programación utilizada para analizar datos, desarrollo de algoritmos y creación de modelos. Utiliza un lenguaje de programación que expresa matemáticas de matrices y arreglos directamente. MATLAB cuenta con *Toolboxes* que están desarrolladas profesionalmente, probadas y documentadas, además sus aplicaciones permiten ver cómo funcionan diferentes algoritmos. También cuenta con Simulink para aplicar el Diseño Basado en Modelos, que se usa para la simulación multidominio, la generación automática de código y la prueba y verificación de sistemas integrados. En visión de computadora MATLAB ofrece el *Toolbox* de Procesamiento de Imágenes, el *Toolbox* de Visión por

Computadora y el *Toolbox* de LIDAR, además de un conjunto de aplicaciones, algoritmos y redes pre-entrenadas. El *Toolbox* de Visión por Computadora ayuda a identificar errores y defectos en objetos como partes de máquinas, utilizando algoritmos de preprocessamiento de imágenes del *Toolbox* de Procesamiento de Imágenes para mejorar las características. [34]

Las técnicas de aprendizaje profundo normalmente se usan para la detectar defectos o fallas, por medio de aplicaciones como el etiquetador de imágenes, videos o LIDAR de MATLAB, permitiendo la creación de máscaras de segmentación para datos de entrenamiento. La detección y rastreo de objetos, una aplicación esencial en visión por computadora, se puede lograr utilizando el Diseñador de Redes Profundas de MATLAB. Además, dentro del *Toolbox* de Visión por Computadora se puede realizar tareas como localización, mapeo mediante visual SLAM, modelización 3D de objetos a través de SFM y conteo de objetos. Las aplicaciones de MATLAB, como *Image Segmenter* e *Image Region Analyzer*, ofrecen una interfaz interactiva para la segmentación y el conteo de objetos en imágenes, lo que lo hace de fácil acceso. [34]

CAFFE

Es un marco de trabajo o *framework* de aprendizaje profundo diseñado con el objetivo de lograr expresión, velocidad y modularidad. Fue desarrollado por Yangqing Jia y contribuyentes de la comunidad, durante su doctorado en UC Berkeley, bajo el *Berkeley AI Research (BAIR)*. En este marco de trabajo los modelos y la optimización se definen mediante configuraciones, de tal manera que no es indispensable codificar. Además, permite cambiar entre CPU y GPU configurando un solo indicador para entrenar una máquina GPU y luego implementar en clústeres o dispositivos móviles. Dada la sencillez de su configuración CAFFE se vuelve perfecto para experimentos de investigación e implementaciones industriales. También, tiene la capacidad para procesar más de 60 millones de imágenes al día con una sola GPU NVIDIA K40 en determinadas condiciones. Lo anterior implica 1 ms/imagen para inferencia y 4 ms/imagen para aprendizaje. [35]

6.4. Protocolos de comunicación

6.4.1. Serial

La comunicación serial es un método sumamente importante en la transferencia de datos entre dispositivos electrónicos, en este caso la información se transmite secuencialmente en un solo canal de comunicación. A diferencia de la comunicación paralela, que transfiere múltiples bits simultáneamente, la comunicación serial envía un bit a la vez, lo que la hace más eficiente en términos de costos y tiempo a nivel de instalaciones. En el funcionamiento de la comunicación serial la transmisión de datos sucede en forma de bits secuenciales a través de un solo canal de comunicación, donde se utilizan diferentes protocolos específicos para establecer la sincronización y el formato de los datos transmitidos. Los datos se envían en serie, uno tras otro, siguiendo un orden predefinido, lo que facilita la interpretación y reconstrucción de la información en el dispositivo que recibe la información. Algunos protocolos son RS-232, RS-485, UART, SPI y I2C, con sus propias características y aplicaciones

específicas en la transmisión de datos. La velocidad de transmisión, medida en baudios, determina la cantidad de bits que se pueden enviar por segundo. Esta influye en la eficiencia de transferencia de datos entre dispositivos, lo cual es un factor determinante para garantizar una comunicación fluida y sin errores. Además, también se deben configurar parámetros de comunicación, como el bit de parada, la paridad y el control de flujo. También es importante tomar en cuenta la distancia de transmisión, dado que puede variar según el tipo de protocolo utilizado y la calidad de la conexión. Algunos protocolos, como RS-232, son adecuados para distancias cortas, mientras que otros, como RS-485d, son más apropiados para distancias más largas. [36]

6.4.2. *Bluetooth*

La comunicación *Bluetooth* es un estándar inalámbrico utilizado para transmitir datos entre dispositivos electrónicos a corta distancia. Esta tecnología permite la conexión y comunicación entre dispositivos como teléfonos móviles, computadoras, auriculares, altavoces y otros dispositivos inteligentes de forma inalámbrica. Esta comunicación es posible gracias al uso de ondas de radio de corto alcance para establecer dicha conexión de manera segura y confiable, facilitando la transferencia de datos sin necesidad de cables físicos. El funcionamiento es realizado a partir de la emisión de señales de radiofrecuencia en una banda de frecuencia específica para establecer una conexión entre dispositivos compatibles, una vez que dos dispositivos están dentro del rango de alcance, no más de unos cuantos metros. Luego, estos tienen la posibilidad de emparejarse y comunicarse entre sí bidireccionalmente intercambiando datos a través de paquetes de información, lo que permite la transferencia de archivos, la reproducción de audio, el control remoto y otras funciones de comunicación. Entre las ventajas de dicho estándar de comunicación, están su versatilidad y bajo consumo de energía, por lo cual es ideal para dispositivos portátiles y de bajo consumo. Además, *Bluetooth* ofrece diferentes perfiles de comunicación que definen cómo los dispositivos interactúan entre sí, como el perfil de manos libres para dispositivos de audio, el perfil de transferencia de archivos para compartir archivos y el perfil de red personal para compartir conexión a Internet. Además, la seguridad es otro aspecto fundamental, este estándar cuenta con protocolos de cifrado y autenticación para proteger la privacidad y la integridad de los datos transmitidos entre dispositivos, como el requerimiento de claves de acceso para establecer conexiones. Por último, las actualizaciones continuas en los estándares de *Bluetooth* han mejorado la seguridad y la eficiencia de la comunicación inalámbrica, garantizando una experiencia de usuario segura y confiable. [37]

6.5. Sistemas operativos en tiempo real

En primer lugar los sistemas operativos de propósito general (GPOS, por sus siglas en inglés), son aquellos que están diseñados para priorizar la interacción entre usuario y el rendimiento antes que un comportamiento determinístico. El comportamiento determinístico, se refiere a la habilidad de un sistema para realizar tareas en tiempos específicos. Ahora bien, los sistemas en tiempo real (RTOS, por sus siglas en inglés), priorizan la realización de tareas en tiempos establecidos, por lo que se aseguran que las tareas con prioridad alta sean ejecutadas. Además, para lograr esto, los RTOS utilizan la multitarea, por medio del

aislamiento temporal y espacial (memoria), de esta manera, para lograr la comunicación entre tareas. Sin embargo para que se puedan comunicar los datos dentro de las tareas, se requiere una estructura de sincronización para intercambio de información, algunos ejemplos son: colas de mensajes, utilizando la estructura FIFO (*First in, first out*), que asegura la lectura de datos en el orden en que fueron enviados. También, se utilizan banderas de eventos, de tal manera que las tareas esperan que se realicen otras tareas, y por último, utilizando memoria compartida, así, las tareas acceden a una región en común, sin embargo esta última estructura es más propensa a la corrupción de datos. En este caso, la forma más sencilla de utilizar un sistema en tiempo real con el entorno de desarrollo de Arduino, es por medio de *FreeRTOS*, el cual se distribuye bajo la licencia de código abierto del *Massachusetts Institute of Technology* (MIT). Esta implementación es sencilla, dado que el núcleo fue diseñado únicamente con tres archivos en lenguaje de programación C y unas cuantas funciones en assembler. [38] [39]

CAPÍTULO 7

Selección de hardware y software

El objetivo de esta selección es analizar las herramientas disponibles para la realización del proyecto, teniendo en cuenta diferentes factores. El primer factor a tomar en cuenta es que se pueda replicar, se busca que el proyecto pueda ser comprobado por futuras generaciones para la mejora continua y posible distribución de una versión mejorada. Luego se toma en cuenta el factor económico, este es importante dado que se busca ayudar a la mayor cantidad de personas posible, entonces se escogerán preferiblemente herramientas de fuente/código abierto. Por último, es necesario tomar en cuenta la disponibilidad de materiales y/o herramientas tanto en cantidad, facilidad de acceso, estabilidad en el mercado, etc.

7.1. Lenguaje de programación

Inicialmente se inicia a trabajar utilizando herramientas presentadas en cursos de robótica, en este caso, se utiliza Python3.10 como lenguaje de programación. Dicha selección, resulta de la compatibilidad que tiene con *OpenCV*, una librería utilizada para poder realizar aplicaciones de visión por computadora, que funciona tanto para leer las imágenes de entrenamiento a utilizar en el modelo de *Machine Learning* como para la captura de imágenes en tiempo real. Es importante mencionar que, para realizar la programación, se optó por utilizar el IDE (Entorno de desarrollo integrado) *PyCharm Community Edition*, dado que cuenta con comandos integrados para manejar entornos virtuales y conexiones con *Git* y *Github*. Además *PyCharm* es uno de los IDE's que cuenta con una fácil incorporación al simulador *Webots*.

7.2. Simulador

En este caso se optó por utilizar un software de código abierto, de igual manera que la librería *OpenCV*. Este simulador es conocido como *Webots*, este simulador cuenta con

diferentes opciones de ambientes y objetos para personalizar. Dicha personalización se refiere a las características espaciales y físicas, además de contar con diferentes agentes robóticos previamente incluidos. Dichos agentes, poseen el modelo virtual, así como la capacidad de controlarlos por medio de controladores en diferentes lenguajes de programación: C, C++, Java, Python, etc.

7.3. Sensor óptico

En este caso se tomó en cuenta el uso de la cámara incorporada en una laptop, dado que cualquier persona que esté trabajando en una, puede utilizar dicha herramienta, ahorrando así la compra de un módulo externo. Dicho sistema de cámara integrada puede variar su resolución desde los 0.4 (848×480) Megapíxeles hasta los 2.1(1920×1080). Este módulo se encuentra integrado en una laptop *Lenovo IdeaPad Flex 5 16IAU7* y es fabricado por la empresa *SunplusIT*, la cual es una empresa líder en la manufactura y distribución de chips para multimedia y aplicaciones automotrices, así como el proveedor de cámaras integradas para la empresa *Lenovo*, por lo cual la replicabilidad de las secciones realizadas con dicho sensor es alta.

7.4. Agente robótico

Se seleccionó el Pololu 3pi dado que es una plataforma robótica versátil y compacta, además es conocida por su gran desempeño en competencias de seguimiento de líneas y resolución de laberintos. Posee un diámetro de 9.5 cm y un peso de 83 g (sin baterías), está diseñado para ser ágil, lo que le permite navegar a través de trayectos intrincados y espacios reducidos con facilidad. También, su pequeño tamaño lo hace ideal para competencias que requieren movimientos precisos y respuestas rápidas a diversos desafíos. El Pololu 3pi posee sensores de seguimiento de líneas, que proporcionan la retroalimentación necesaria para que ajuste sus movimientos y se mantenga en el curso, demostrando sus capacidades en tareas que exigen precisión y consistencia. Además, está equipado con un regulador buck de Pololu, este es regulador de conmutación síncrono reductor que es capaz de reducir de manera eficiente voltajes de entrada de hasta 50 V a una salida estable de 3.3 V, asegurando una distribución de energía confiable a los componentes del robot. Además, la página oficial cuenta con tutoriales de video que demuestra el ensamblaje del robot 3pi+ 32U4, una versión más reciente del robot 3pi. Todas estas características fueron tomadas en cuenta para la selección del robot móvil a utilizar, sumado a que es uno de los robots con capacidad de desplazamiento presentes en la Universidad del Valle de Guatemala, lo que facilita la disponibilidad para realización de pruebas. [40]

CAPÍTULO 8

Bases de datos

Como cualquier modelo de aprendizaje de máquina, es necesario tener una base de datos con la cuál dicho modelo sea capaz de detectar las características deseadas por el usuario. En este caso, se busca iniciar con bases de datos existentes, las cuales posean fotografías de rostros en diferentes posiciones/gestos. Sin embargo, también es valioso el poder generar una base de datos propia, de tal manera que se pueda disminuir el porcentaje de error en las tareas realizadas. También se detalla la variación en la captura de datos para mejorar la predicción de los modelos. Además, también se contempla la utilización de aprendizaje supervisado, por lo que el proceso de clasificación de manera manual, se debe realizar para todos los datos obtenidos.

8.1. ICPR (Grupo de modelos 1)

Al inicio se realizó una búsqueda de bases de datos que tuvieran diferentes rostros posicionados en distintos ángulos. Sin embargo, hubo una única base de datos pública publicada por la *ICPR (International Workshop on Visual Observation of Deictic Gestures)*, en Cambridge, Reino Unido, con 2790 imágenes disponibles para cualquier propósito. En esta base de datos cuenta con 15 sujetos de prueba, a cada sujeto se le realizaron dos pruebas y en cada prueba se realizó un total de 93 fotografías. El grupo de imágenes se obtuvieron al mezclar los siguientes ángulos: 7 de inclinación (vertical) y 13 panorámicos (horizontal) además de una dos fotografías extra, una en la posición completamente hacia arriba y otra completamente hacia abajo. Las imágenes en esta base de datos se encuentran en un formato JPG, además cada imagen cuenta con un documento de texto que contiene: el nombre del archivo y el centro en el eje X, en el eje Y, el alto y el ancho de la cara correspondiente. [41]

Las fotografías de la base de datos se encuentran nombradas en primer lugar por su ángulo de panorámica y luego por su ángulo de inclinación, por lo que se puede observar que para cada sujeto las fotografías aparecen desde la panorámica extrema derecha y la inclinación extrema inferior y terminan en los extremos opuestos. Por lo que la fotografías recorren

primero los ángulos de panorámica y luego incrementan los ángulos de inclinación. Además, estas fotografías no cuentan con algún archivo de etiquetas, por lo que estas se asignaron manualmente, tomando en cuenta el orden previamente explicado en que se encuentran ordenadas. A continuación, se muestra el orden mencionado anteriormente en la Figura 11:

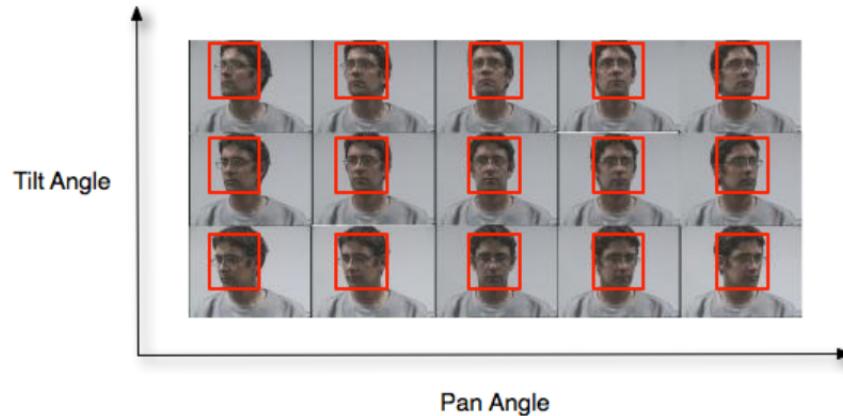


Figura 11: Muestra de 15 fotografías del sujeto 12. [41]

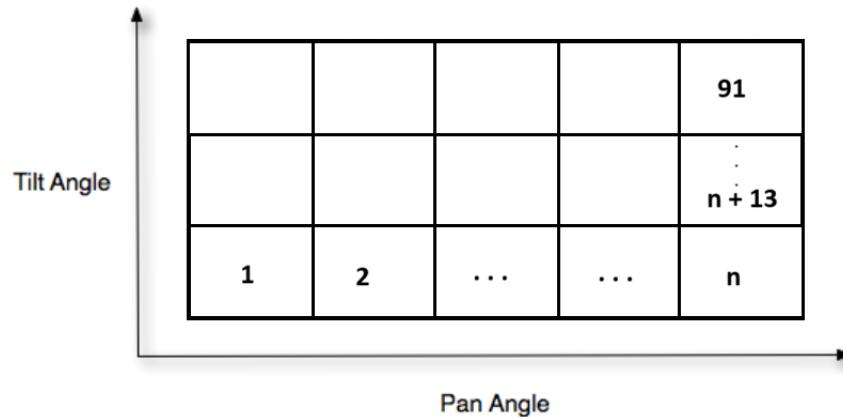


Figura 12: Orden de guardado para cada serie de fotografías. (Generado por Gerardo Fuentes)

Además de este orden, todas las series de fotografías iniciaban con la imagen de la cabeza completamente hacia arriba y finalizaban con la posición completamente abajo, de frente para ambas. Dado que se tenían 15 sujetos de prueba, se procedió a guardar los archivos JPG de 11 (2046 imágenes) sujetos en un directorio de entrenamiento y los 4 (744 imágenes) restantes en el de pruebas. Al tener en cuenta el orden de los archivos y la cantidad en cada directorio se procedió a realizar las etiquetas por medio de un documento *Excel* con formato *.xlsx*. En dicho documento se creó una hoja para cada grupo de etiquetas, esto quiere decir, una hoja para etiquetas de entrenamiento y otra para etiquetas de prueba. Por lo que, en cada hoja se puede encontrar una columna con las etiquetas en el orden establecido en la Figura 12.

8.1.1. Etiquetas para clasificación

Para la primera experimentación se consideró clasificar las imágenes en 9 clases diferentes, de tal manera que la orientación de la cabeza respecto de la cámara funcionara de manera similar a una palanca de mandos analógica. A continuación, en la Figura 13, se presenta una visualización de dicha clasificación:

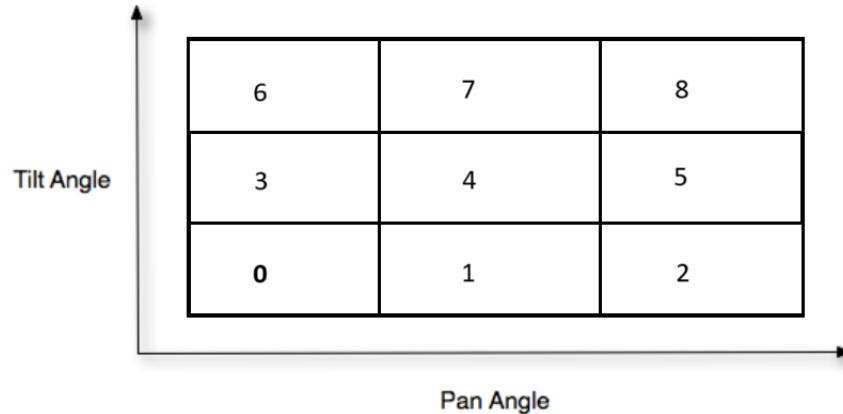


Figura 13: Clasificación de imágenes en 9 etiquetas. (Generado por Gerardo Fuentes)

Utilizando esta cantidad de clases, se realizaron dos formas de etiquetado en las imágenes, en las cuales solo varía la cantidad de fotografías que fueron etiquetadas dentro de una clase u otra. En las Figuras 14 y 15 se podrá observar que la dimensión de cada celda está representada con $(n \times m)$, donde n corresponde a la cantidad de ángulos panorámicos que abarca y m la cantidad de ángulos de inclinación (según las muestras tomadas del ICPR). Además, la posición de las celdas en las gráficas si corresponde a la numeración de la Figura 13.

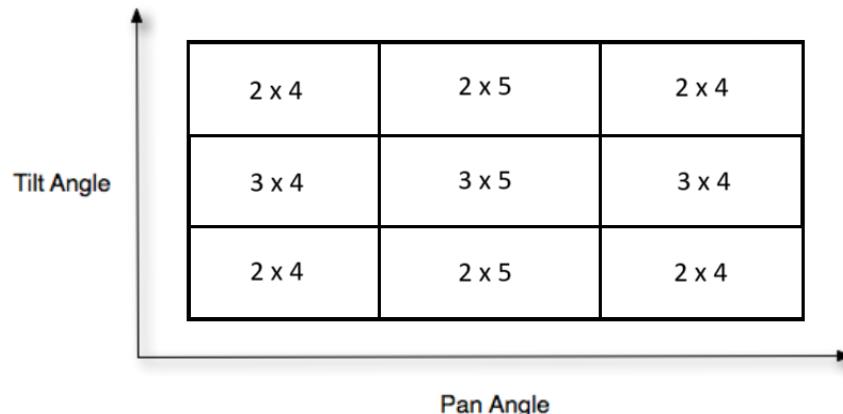


Figura 14: Primera distribución de etiquetas. (Generado por Gerardo Fuentes)

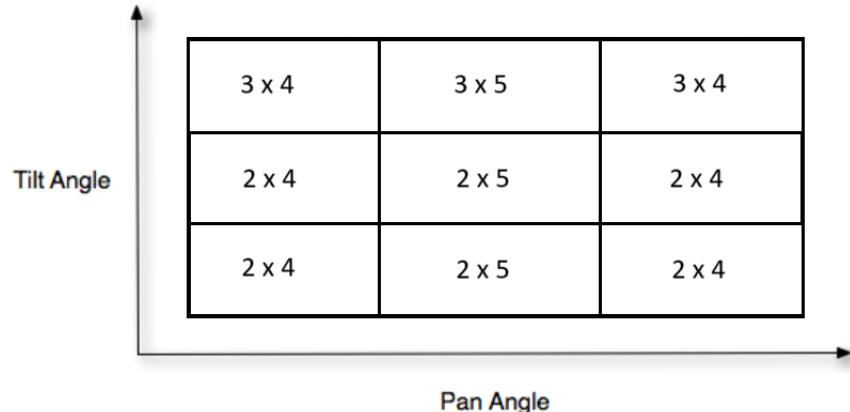


Figura 15: Segunda distribución de etiquetas. (Generado por Gerardo Fuentes)

Luego de pruebas realizadas con resultados no satisfactorios en la sección de algoritmos para visión por computadora, la siguiente experimentación consistió en reducir la cantidad de clases a 6. En este caso para realizar un movimiento hacia atrás, se requiere un giro completo, en vez del sistema de joystick propuesto anteriormente. La división de clases se muestra a en las Figuras 16, 17 y 18:

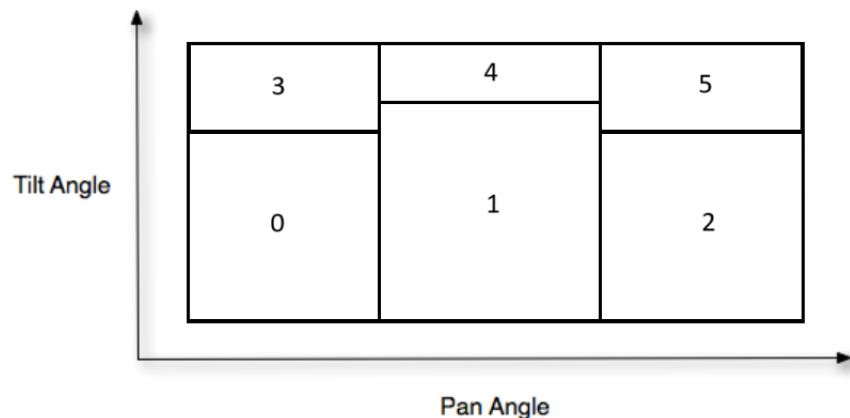


Figura 16: Clasificación de imágenes en 6 etiquetas. (Generado por Gerardo Fuentes)

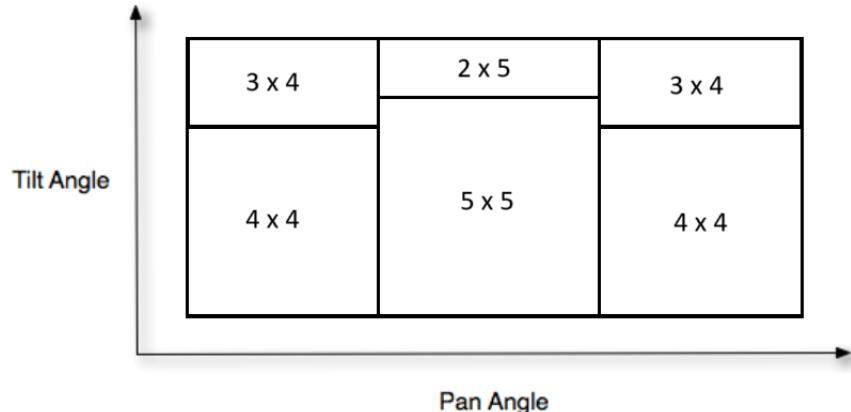


Figura 17: Tercera distribución de etiquetas. (Generado por Gerardo Fuentes)

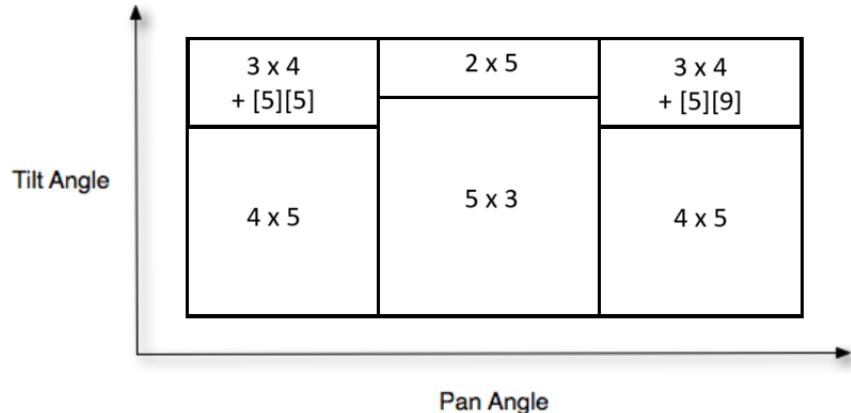


Figura 18: Cuarta distribución de etiquetas (Las posiciones se cuentan desde la esquina inferior izquierda). (Generado por Gerardo Fuentes)

8.2. ICPy y fotografías personales (Grupo de modelos 2 y 3)

Dado que los resultados con los modelos de reconocimiento de orientación iniciales no fueron los esperados en ningún caso, se procedió a realizar 2 series de fotografías personales con la cámara integrada. Cada serie consta de 9 fotografías con posiciones que simulan las 9 clases inicialmente definidas a una distancia de 45 cm aproximadamente. Las series fueron capturadas en un salón de laboratorio del Departamento de Ingeniería Electrónica, Mecatrónica y Biomédica de la Universidad del Valle de Guatemala y en espacios diversos de la casa de Gerardo Andres Fuentes Bámaca (autor). Estas series se agregan al set de entrenamiento, sin embargo, también se requerirá realizar una disminución de clases nuevamente. La nueva cantidad para la primera es de 4, y se tomarán en cuenta la posición de la cabeza en estado neutro, a los lados y hacia arriba o abajo. Luego de obtener el resultado, se realizará una nueva prueba de etiquetas con únicamente 3 clases, al centro y a los lados, de no obtener resultados satisfactorios. Estas series se agregarán a la base de datos utilizada para entrenar

los primeros modelos. En la Figura 19, se observa un ejemplo con la posición de la cabeza hacia arriba.



Figura 19: Captura de fotografías simples. (Generado por Gerardo Fuentes)

8.2.1. Ajuste

Dado que los resultados aún no fueron óptimos en el segundo grupo de modelos, se procedió a utilizar un apoyo visual. En esta caso se utilizaron marcadores en forma de puntos verdes que se colocaron en la frente, mentón, y pómulos de los sujetos de prueba, como se puede observar en la Figura 20. En esta serie de datos, se obtuvieron datos de 10 sujetos, que voluntariamente se prestaron para la realización de las series de fotografías determinadas en pruebas anteriores con Gerardo Fuentes (autor). Con estas series de fotografías se obtuvieron 968 imágenes que también se clasificaron en 70 % para entrenamiento y 30 % para validación, además de la base de datos ICPR.



Figura 20: Captura de fotografías con marcadores físicos. (Generado por Gerardo Fuentes)

8.3. Únicamente fotografías personales (Grupo de modelos 4)

Luego de que los resultados obtenidos con los grupos de modelos de aprendizaje 2 y 3, no fueran los esperados, se procedió a realizar un cambio en el enfoque de obtención de datos y modelos de aprendizaje. Tomando esta decisión en cuenta, también se decidió trabajar con fotografías realizadas únicamente en una configuración inicial para el usuario. Sin embargo, para entrenar el modelo de una manera efectiva, se requirió realizar un algoritmo que capturara 525 imágenes por posición de la cabeza establecida, en dicha configuración inicial. En este caso se probó este nuevo algoritmo con 4 posiciones/gestos: derecha, centro, izquierda, junto con un gesto de boca abierta para un caso y un gesto de sonrisa para otro caso. Luego se realizaron pruebas con 4 posiciones/gestos: derecha, centro, arriba, izquierda, después se realizó una variante con 5: arriba, abajo, centro, derecha e izquierda. Por último, se realizaron pruebas con las 9 posiciones/gestos iniciales. Además, por la naturaleza del nuevo algoritmo, no se requiere tanta variación en las condiciones externas al capturar las fotos, en este caso únicamente se requiere que la iluminación trasera a la cabeza del usuario no sea mucho mayor que la frontal.

CAPÍTULO 9

Algoritmos para visión por computadora

En esta sección, se detalla el proceso realizado a nivel de software para poder utilizar los modelos de aprendizaje de máquina que utilizarán visión por computadora. Entre estas tareas, se detallan los procesos realizados con las fotografías de las bases de datos obtenidas, externas y personales. Asimismo, se muestra la selección de hiperparámetros para la generación de modelos predictivos así como los resultados teóricos de cada uno de los modelos propuestos. Por último, se describe un método para realizar las tareas de manera más eficiente, la programación multi-hilos, que pretende realizar tareas en paralelo aprovechando los múltiples núcleos del procesador, en vez de realizar tareas de forma secuencial.

9.1. Grupos de modelos: 1 a 3

9.1.1. Preprocesado de imágenes

Inicialmente se diseñó un código para extraer fotografías desde un directorio, etiquetas desde un documento con extensión *.xlsx* y luego transformarlos en matrices de datos que para poder ser utilizados en los procesos de *Machine Learning*. Para la realización de este código, se utilizaron las siguientes librerías: *OS*, *CV2 (OpenCV)*, *Numpy* y *Pandas*. Además de esto el proceso para la realización del código fue el siguiente:

Transformación de fotografías

En este caso, se utilizó la librería *OS* para crear una lista con nombres en formato cadena de los archivos en un directorio dado. Luego, utilizando la librería *Numpy*, se creó una matriz de ceros para guardar todas las fotografías convertidas en formato de matriz. Para realizar la conversión, previamente se realizó una prueba para obtener las dimensiones mínimas con las que puede trabajar la cámara integrada con la librería *CV2*, cuyo tamaño es 320×180 píxeles. Después, dentro de un bucle *for* se utilizó la librería *CV2* obtener la matriz de pixeles

de cada fotografía, ajustar el tamaño a las dimensiones encontradas y luego transformar el canal de color por defecto BGR (*Blue, Green, Red*) a RGB (*Red, Green, Blue*).

Transformación de etiquetas

Para transformar las etiquetas en matrices de datos, únicamente se requirió la librería *Pandas*, ya que nos permite utilizar una función específica para leer documentos con formato *.xlsx*, así como acceder a las hojas y columnas del documento. Al guardar el resultado de esta función el siguiente paso es convertirlo a un formato de matriz con funciones de *Numpy*.

Exportación de datos finales

Antes de realizar la exportación de datos, es importante aplicar una transformación de tipo flotante a entero, en las matrices, y de esta manera se ahorra espacio en el disco duro. Ahora bien, la librería *Numpy* cuenta con una función para exportar matrices de datos de manera individual (una variable a la vez), o bien agruparlas en un solo archivo. En este caso, dado que se necesitan realizar diferentes pruebas, se estará exportando de manera individual, en cambio para los resultados finales, se utilizará la función de guardar de manera agrupada.

Filtrado de imágenes (Modelo No. 8)

Luego de las pruebas con los primeros 7 modelos de reconocimiento facial, dónde únicamente se utilizaban los datos de las fotografías a un tamaño de 320×180 píxeles, con canal RGB, para alimentar el proceso, se decidió utilizar la librería *OpenCV* para aplicar filtros de bordes a las fotografías, de tal manera que el modelo de aprendizaje no tome en cuenta los colores sino únicamente los bordes de los rostros. Este proceso se aplicó en la sección del código para transformar las fotografías, y en la sección que recopila la imagen en tiempo real.

9.1.2. Entrenamiento de modelos de *machine learning*

Para las primeras pruebas se utilizó el framework *TensorFlow* y *Keras* para poder realizar los diferentes modelos de *Machine Learning* para reconocimiento de orientación de cabeza, utilizando las matrices de datos obtenidas a partir de las bases de datos recopiladas. Para este código se utilizaron las librerías *TensorFlow*, *Matplotlib*, *Numpy* y *Seaborn*. El código fue realizado de la siguiente manera:

Extracción de datos previamente procesados

Luego de exportar los archivos transformados de la base de datos, se generan archivos con extensión *.npy* para los archivos de una variable y *.npz* para los de múltiple variables. Dichos archivos pueden ser importados utilizando también funciones de la librería *Numpy*, para guardarlo en variables utilizables dentro del código, optimizando así el tiempo de

transformación de datos. El siguiente paso fue normalizar los valores de las matrices, dado que son imágenes RGB, cada píxel de estas está representado por 3 valores que varían entre 0 y 255, y se necesitan valores entre 0 y 1, por lo que únicamente se tuvo que dividir tanto la variable con datos de entrenamiento y prueba entre el valor 255.

Grupo de aprendizaje 1

Inicialmente, dado que se trata de un modelo que trabaja con matrices derivadas de fotografías, se utilizó una CNN (Red Convolutacional Neuronal), la cual permite detectar patrones en las imágenes, siendo esta una aplicación de clasificación. Utilizando *Tensorflow* y *Keras*, se definieron 7 capas para la generación del modelo. En este caso se utilizó en primera instancia una capa convolucional en 2D, con 10 kernels de tamaño 3×3 , función de activación *ReLU*, y por ser la primera capa se agregan las dimensiones de entrada (fotografía). Luego, se aplicó una capa con la técnica de *Max Pooling* con tamaño 2×2 . Después, se aplica una capa de aplanado previa necesaria para poder utilizar las siguientes capas densas. Esta vez se utilizaron 3 capas densas, con 75, 150 y 75 nodos respectivamente con función de activación *ReLU*. Por último, se aplica una capa densa con una cantidad de nodos definida por el número de clases a utilizar y función de activación *Softmax*. Con esta especificación de hiperparámetros, se procede a unir las capas en un modelo secuencial, para luego compilarlo con un optimizador *Adam*, para la pérdida se utilizó *Sparse categorical crossentropy*, dado que las etiquetas fueron definidas como enteros y no en el formato *One-Hot*. Para finalizar con los hiperparámetros, en la opción métricas se escoge la opción *Accuracy* o precisión. Luego de entrenado el modelo, se procede a guardarla con la función de guardar en *Keras*, en el cual el argumento es el directorio donde se desea guardar el archivo con extensión *.keras* y poder utilizarlo en otros códigos.

Al tener configurado el modelo, se procede al entrenarlo, guardarla y exportar las gráficas descriptivas respectivas. En esta etapa se cuenta con 4 modelos que varían dado que se utilizó cada uno de los diferentes grupos de etiquetas. En las Figuras 21 - 28 se puede observar que se produce un sobreajuste, tanto en pérdida como en exactitud en todos los modelos entrenados. Los valores en pérdida de validación no disminuyen más allá del valor 1.0 y la exactitud en validación no supera el 70 %. Además, en las matrices de confusión, de todos los modelos, se observa que la predicción falla al detectar la posición vertical. Esto sucede dado que las fotografías de la base de datos ICPR, tienen el mismo fondo en todos su casos. Además, la mayoría de los sujetos de prueba son de sexo masculino, caucásicos y con un rango de edad similar. Además de ser únicamente 11 personas diferentes.

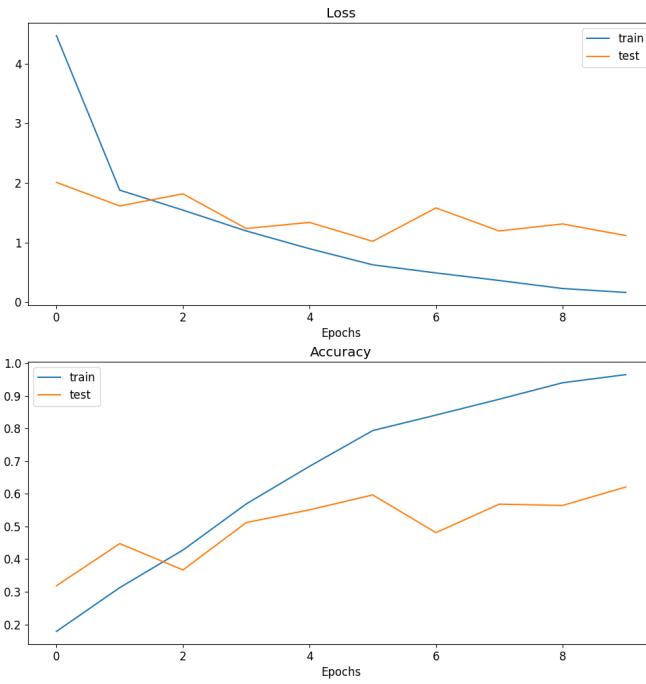


Figura 21: Pérdida y exactitud del primer modelo con 9 clases. (Generado por Gerardo Fuentes)

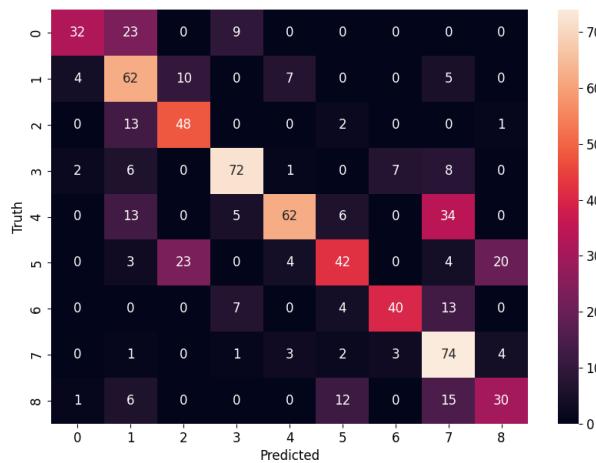


Figura 22: Matriz de confusión del primer modelo con 9 clases. (Generado por Gerardo Fuentes)

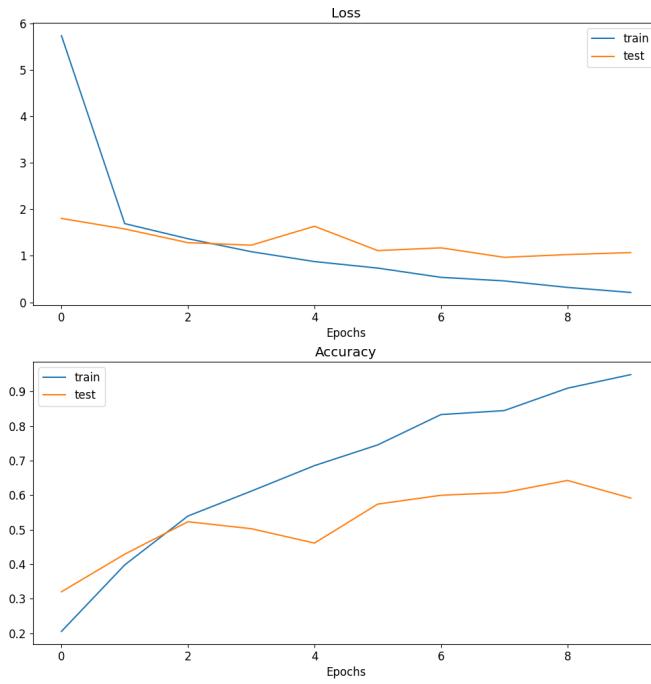


Figura 23: Pérdida y exactitud del segundo modelo con 9 clases. (Generado por Gerardo Fuentes)

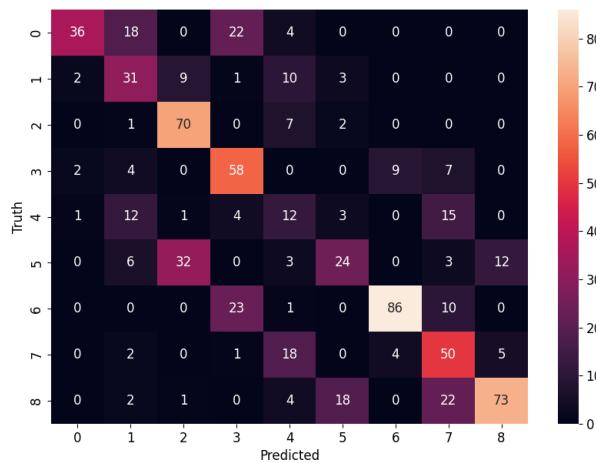


Figura 24: Matriz de confusión del segundo modelo con 9 clases. (Generado por Gerardo Fuentes)

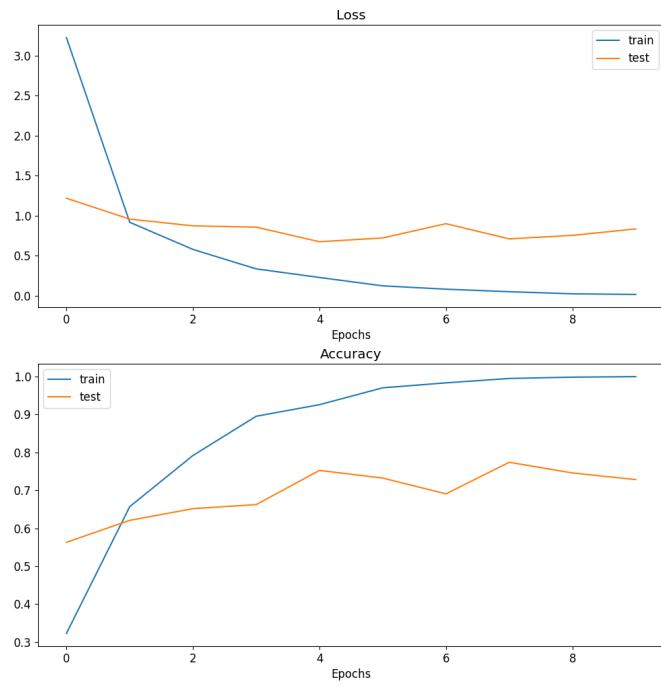


Figura 25: Pérdida y exactitud del primer modelo con 6 clases. (Generado por Gerardo Fuentes)

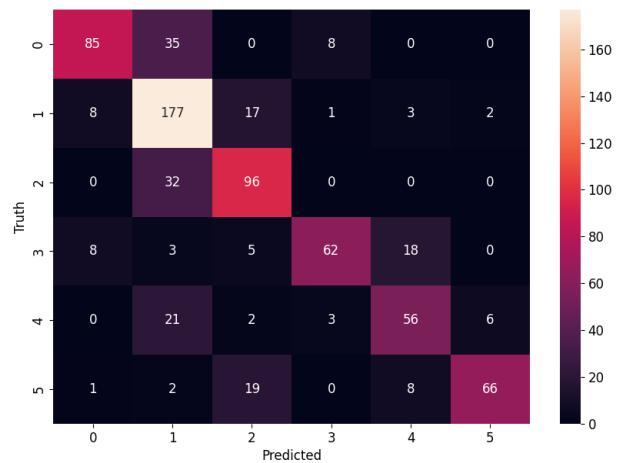


Figura 26: Matriz de confusión del primer modelo con 6 clases. (Generado por Gerardo Fuentes)

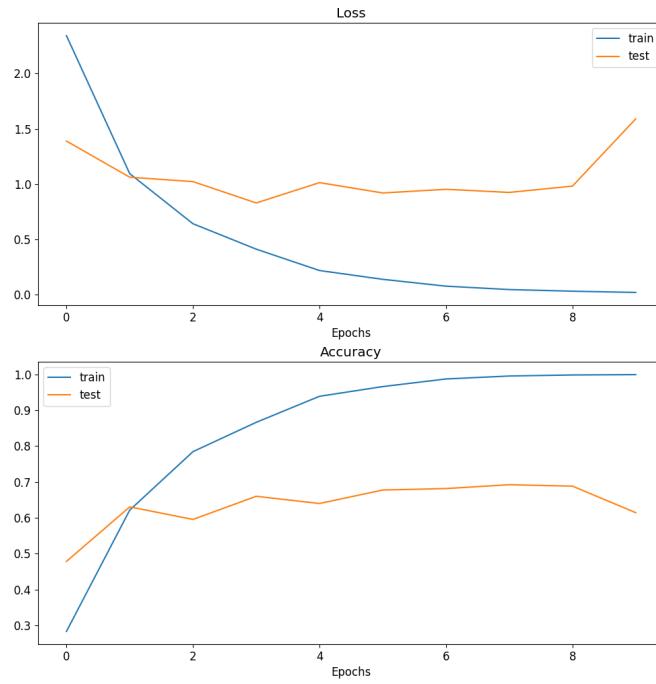


Figura 27: Pérdida y exactitud del segundo modelo con 6 clases. (Generado por Gerardo Fuentes)

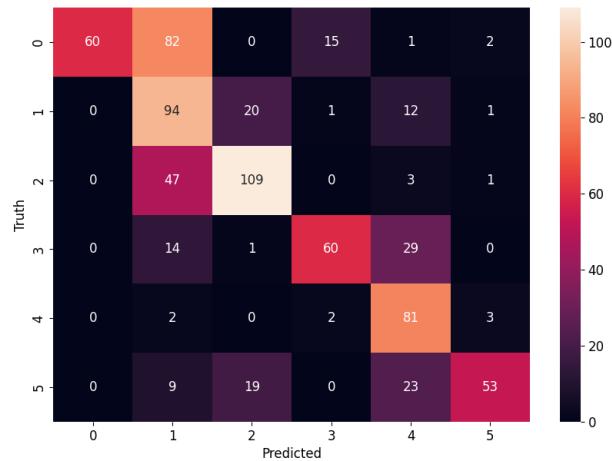


Figura 28: Matriz de confusión del segundo modelo con 6 clases. (Generado por Gerardo Fuentes)

Grupo de aprendizaje 2

En este caso se utilizó la misma configuración para el modelo de aprendizaje utilizado en los primeros 4 modelos, por lo que únicamente se varió la cantidad de etiquetas utilizadas. Al observar la Figura 29, se resalta una disminución en el valor de pérdida al realizar la validación, que anteriormente convergía aproximadamente en los valores de 1.0 y ahora se acerca a 0.5. En la gráfica de exactitud, por primera vez se logró superar el 80 % en validación. Además, el comportamiento de la exactitud se aprecia de manera visible en la matriz de confusión en la Figura 30.

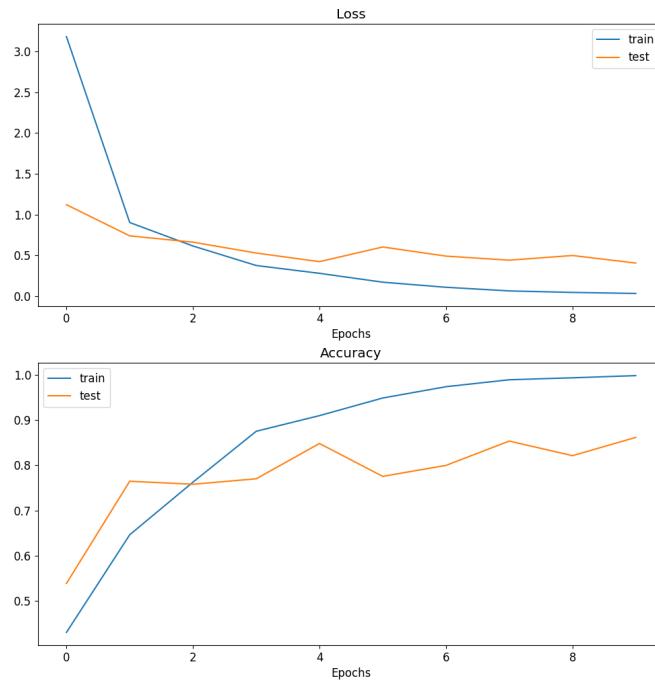


Figura 29: Pérdida y exactitud del modelo con 4 clases. (Generado por Gerardo Fuentes)

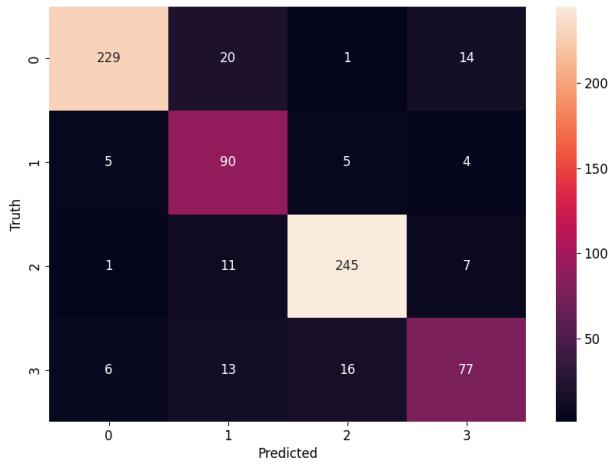


Figura 30: Matriz de confusión del modelo con 4 clases. (Generado por Gerardo Fuentes)

En la Figura 31 se puede observar que la pérdida fluctúa entre los valores 0.5 y 0.1, lo cual es muy similar al modelo de 4 etiquetas. En las gráficas de exactitud, se observa que los datos predichos nuevamente generan una exactitud levemente superior al 80 %. Por otro lado, en la matriz de confusión en la Figura 32, se observa que las magnitudes de los aciertos son similares entre sí respecto de las clases. En general el modelo de 3 clases tiene una buena base teórica para funcionar con el usuario Gerardo Fuentes, sin embargo utilizar 4 clases resulta más útil a nivel de comandos.

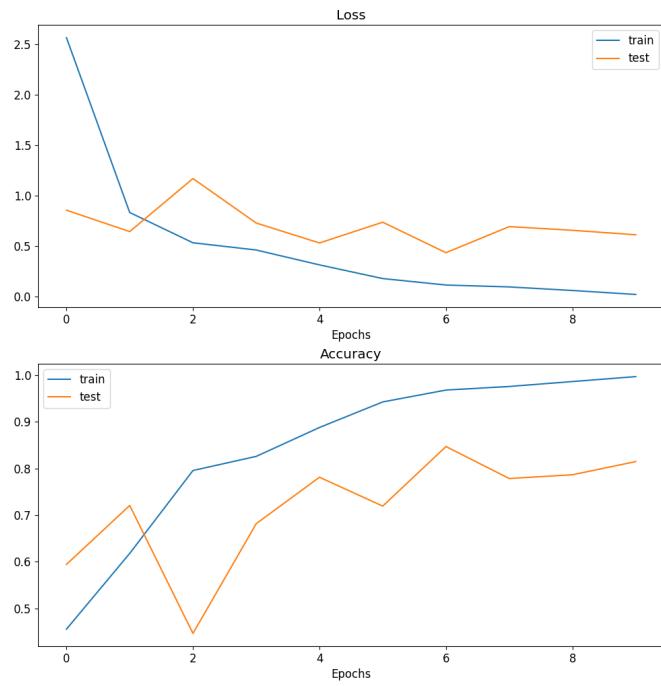


Figura 31: Pérdida y exactitud del modelo con 4 clases. (Generado por Gerardo Fuentes)

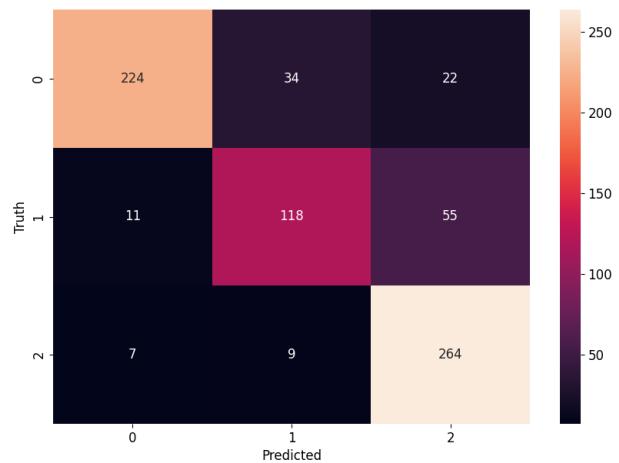


Figura 32: Matriz de confusión del modelo con 4 clases. (Generado por Gerardo Fuentes)

Grupo de aprendizaje 3

En este grupo de modelos, se utiliza la misma configuración para la red neuronal que en los modelos anteriores. Sin embargo, en este caso se incrementó la cantidad de datos para entrenamiento, se utilizan 4 clases en todos los modelos, el octavo modelo utiliza filtros y el noveno fue cargado con imágenes de sujetos de prueba que utilizan marcadores faciales, los cuales se explican en la sección de bases de datos y fotografías personales. En la Figura 33 se puede observar que los valores de pérdida por primera vez lograron valores menores a 1.0, y la exactitud logra superar el 80 % de aciertos. Además, la matriz de confusión de la Figura 34 muestra una diagonal de aciertos mucho más clara.

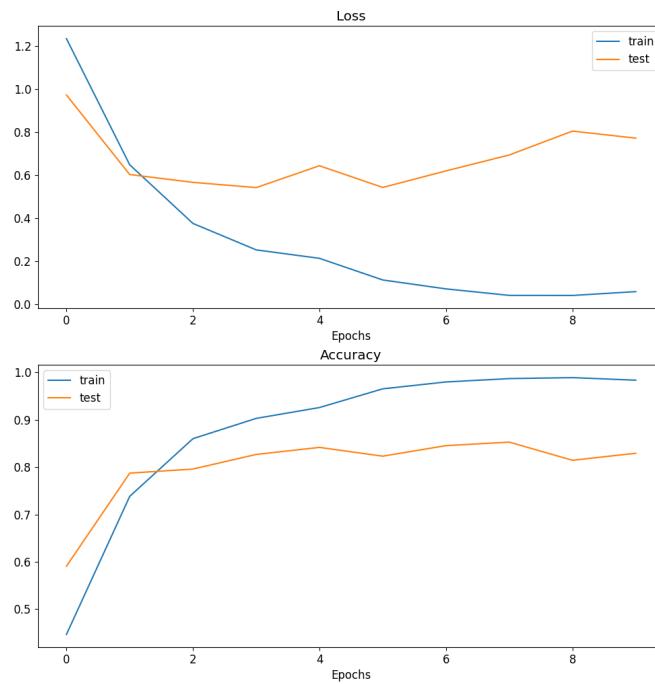


Figura 33: Pérdida y exactitud del modelo con 4 clases y aumento de imágenes de entrenamiento.
(Generado por Gerardo Fuentes)

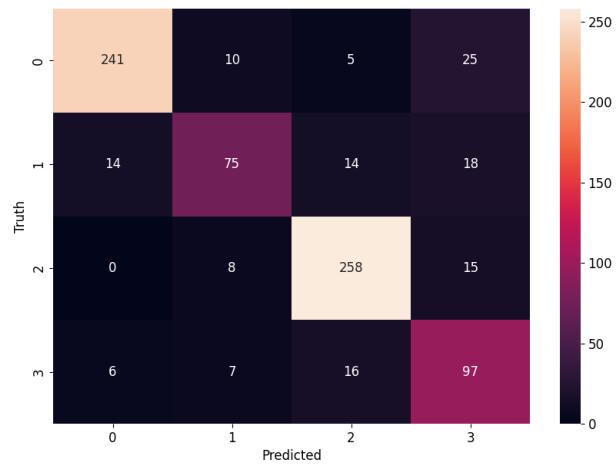


Figura 34: Matriz de confusión del modelo con 4 clases y aumento de imágenes de entrenamiento.
(Generado por Gerardo Fuentes)

A continuación se observa el octavo modelo, el cual incluye un filtro de bordes para las fotografías. En la Figura 35, se muestra un claro paso atrás en la disminución de valores de pérdida, superando nuevamente el valor de 1.0, mientras que para la exactitud de aciertos, el porcentaje queda aún menor a 70 %. Este comportamiento erróneo es visible en la matriz de confusión de la Figura 36.

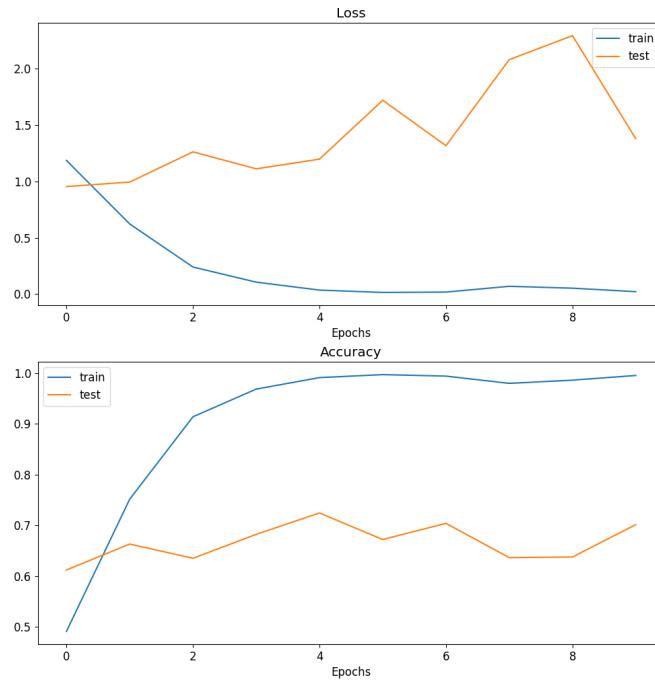


Figura 35: Pérdida y exactitud del modelo con 4 clases con filtro de bordes. (Generado por Gerardo Fuentes)

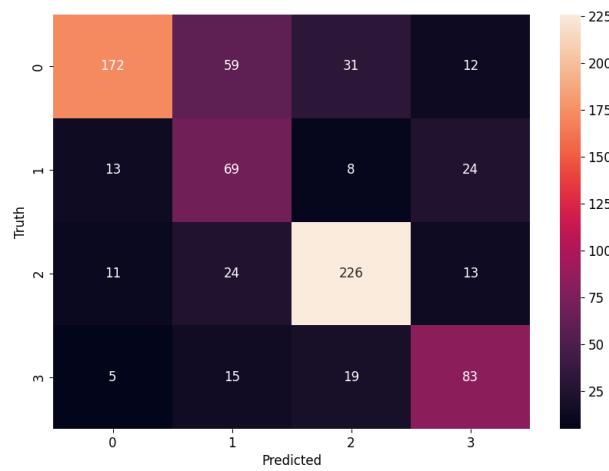


Figura 36: Matriz de confusión del modelo con 4 clases con filtro de bordes. (Generado por Gerardo Fuentes)

Para este noveno modelo, en la Figura 37 el resultado de pérdida nuevamente supera el valor de 1.0, y la exactitud no supera el 70 %. Sin embargo, la matriz de confusión de

la Figura 38 muestra un comportamiento más claro que el octavo modelo en cuanto a la diagonal de aciertos. Esto último tiene que ver con el incremento de imágenes obtenidas de los 10 sujetos de prueba.

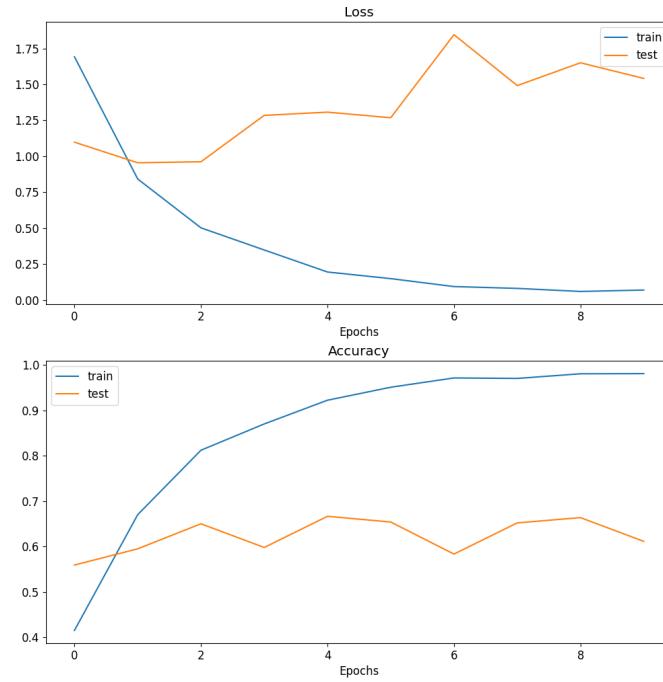


Figura 37: Pérdida y exactitud del modelo con 4 clases y marcadores faciales. (Generado por Gerardo Fuentes)

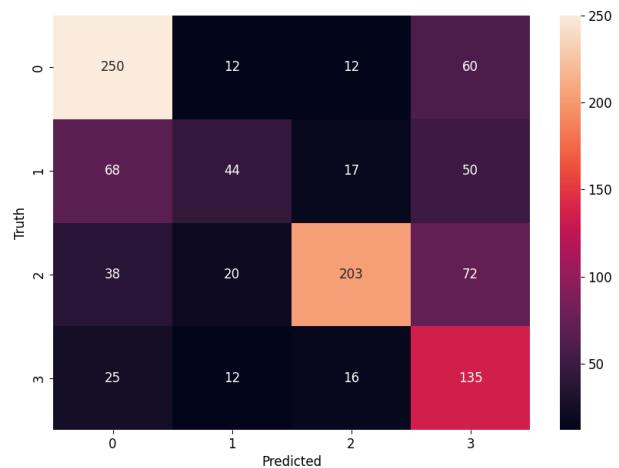


Figura 38: Matriz de confusión del modelo con 4 clases y marcadores faciales. (Generado por Gerardo Fuentes)

9.2. Grupo de modelos 4

9.2.1. Preprocesado de imágenes

Con este nuevo enfoque se utilizó un algoritmo que se ayuda de una librería capaz de trazar puntos de la cara de un usuario para luego guardarlos, luego se generan las etiquetas de cada posición/gesto y se guarda en un documento con extensión *.xlsx*. Por último existe otro código que se encarga de cargar dichos datos y utilizarlos con modelos de *Machine Learning*. Para llevar estas tareas a cabo, se utilizaron las siguientes librerías: *OS*, *CV2 (OpenCV)*, *Numpy*, *Matplotlib*, *Mediapipe*, *Math*, *Time*, *Pygame*, *Seaborn*, *Tensorflow* y *Pandas*. Además de esto el proceso para la realización del código fue el siguiente:

Obtención de trazado de puntos

Es en este punto donde se hace uso de la librería *Mediapipe*, esta es una librería que es capaz de trazar puntos de referencia en las manos, en la cara y en el cuerpo completo, y con ello calcular poses. En este caso, se utilizaron las funciones para obtener los puntos de referencia de la cara. Luego, se realizaron modificaciones para poder obtener los puntos de referencia en forma de matriz de forma ordenada y así poder realizar un grafo que permita verificar las posiciones de dichos puntos. Las Figuras 39 y 40 se obtuvieron a partir de capturas de la cámara utilizando la librería *OpenCV*, como se muestra a continuación:

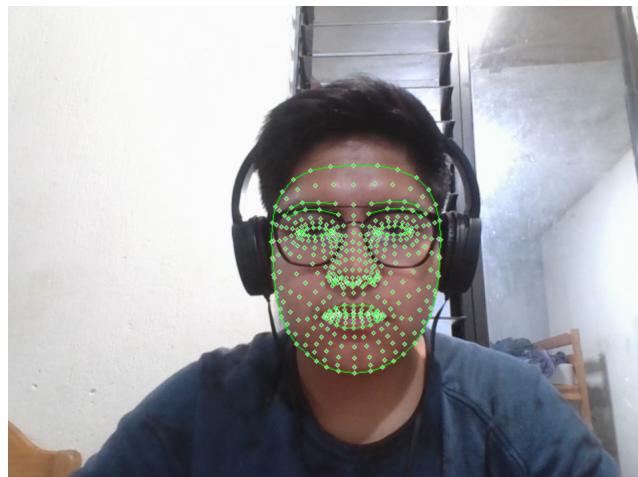


Figura 39: Despliegue de los puntos de referencia desde las funciones de *Mediapipe*. [6]

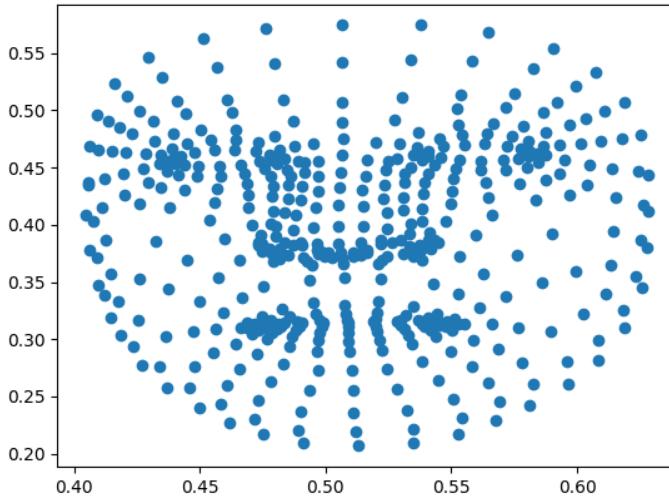


Figura 40: Despliegue de los puntos de referencia desde la matriz obtenida. (Generado por Gerardo Fuentes)

Captura de datos

En esta etapa se realiza una captura de datos ordenada, de tal manera que hay un indicador de audio alertando al usuario para cambiar de posición/gesto. Este indicador consta de tres sonidos con la misma base, con la única diferencia en que cada uno tiene una frecuencia mas alta que el anterior, los cuales fueron obtenidos por medio de una plataforma en línea llamada *sfxr.me* y se encargan de advertir al usuario que prepare su posición/gesto. Luego, se realiza la captura de los 468 puntos de referencia de la posición/gesto del usuario en forma de matriz. Al finalizar la captura de datos, suena una alarma con un sonido completamente diferente, obtenida de la misma plataforma. Este proceso se repite la cantidad de veces que corresponde a la cantidad de clases deseadas en el modelo. Además, el código realiza la captura de imágenes por cantidad, es decir, en este caso se configuró para que se capturen 525 imágenes por posición/gesto. Y para cada posición/gesto, se realiza un proceso en el cual, se revisan los directorios a utilizar, tanto de entrenamiento como validación y así se enumeran. Además, la imagen y la matriz correspondiente se guarda con una etiqueta según dicha posición/gesto y se retienen en una carpeta temporal en formato JPG y NPY.

Guardado de datos

En esta parte, el código funciona cuando el usuario está conforme con los resultados, es entonces cuando se realiza una separación de aproximadamente el 70 % de cada clase (posición/gesto) la cual se guarda en un directorio destinado para los datos de entrenamiento, y luego el 30 % en un directorio de validación. Luego de hacer esta separación, se procede a verificar la etiqueta en el nombre de cada archivo NPY (no se utilizan los archivos JPG), y se crean las etiquetas que se guardan en el archivo con formato XLSX (esto se realiza para

tener una mejor trazabilidad), el cual también separa según archivos de entrenamiento o validación.

Luego existe otro código que se encarga de crear la matriz de adyacencia requerida para las *Graph neural networks (GNN)*. En este caso, se quiere realizar un énfasis en la relación que tienen entre sí los puntos de referencia de los labios, ojos, cejas, y el contorno general de la cara. Para poder identificar los puntos de estas características, se obtuvieron las listas detalladas en pythonrepo.com. Luego, dado que la matriz de adyacencia indica el efecto que tiene un nodo/punto sobre otro, se procedió a realizar una matriz simétrica por medio de ciclos *For* para cada lista en una matriz cuadrada de 468×468 .

9.2.2. Entrenamiento de modelos de *machine learning*

Nuevamente utilizó el framework *TensorFlow y Keras* para poder realizar los diferentes modelos de *Machine Learning* para reconocimiento de orientación de cabeza, utilizando las matrices obtenidas del usuario en la nueva configuración inicial. Para este código se utilizaron las librerías *TensorFlow*, *Matplotlib*, *Os*, *Pandas*, *Numpy* y *Seaborn*. El código fue realizado de la siguiente manera:

Extracción de datos previamente procesados

Esta etapa inicia con la configuración de directorios, se establecen las variables que guardarán las direcciones de los directorios donde fueron guardados los archivos, luego se generan matrices de ceros que tengan las dimensiones necesarias para crear las variables de entrenamiento y validación para etiquetas y datos respectivamente. Para el caso de los datos de entrenamiento y validación, se realizó un ciclo que guardara en cada posición de la matriz, un grupo de puntos de referencia. Ahora bien, para el caso de las etiquetas, bastó con importar las columnas desde el archivo con formato XLSX utilizado para guardar las etiquetas, y luego transformar el tipo de datos a *enteros*. Por último, la matriz de adyacencia (archivo guardado en formato NPY) creada se guardó en cada posición de dos matrices con longitudes igual a la de entrenamiento y validación respectivamente.

Grupo de aprendizaje: 4

Con este tipo de datos, los cuales no provienen de imágenes, sino que de listas de puntos de referencias que pueden formar una gráfica, se optó por buscar un enfoque de Redes Neuronales Gráficas. En este caso, se utilizó una herramienta de *Tensorflow*, que nos permite realizar una operación con estructura similar a las Redes Neuronales Gráficas, dado que este *framework* no cuenta con capas específicas para esta aplicación. Dicha herramienta, es la convolución en una dimensión, la cual comparte las funciones de encontrar conectividad local entre los nodos por los cuales se realiza la operación en una Red Neuronal Gráfica. En este caso el número de clases varía dependiendo de la prueba realizada, se utilizaron 3 capas de convolución en una dimensión con 75, 150 y 75 nodos respectivamente y función de activación *ReLU*. Luego se aplicó la técnica *Global average pooling 1D*, que calcula la media de las características a lo largo de la primera dimensión. Por último, se aplica una capa

densa con una cantidad de nodos definida por el número de clases a utilizar y función de activación *Softmax*. Luego se unen las capas en un modelo secuencial, con un optimizador *Adam*, para la pérdida se utilizó *Sparse categorical crossentropy*, dado que las etiquetas fueron definidas como enteros y no en el formato *One-Hot*. Para finalizar, la opción métricas se fijó en *Accuracy* o precisión. Luego de compilado el modelo con 10 épocas en cada caso, se guarda en un directorio y luego se realizan matrices de confusión y gráficas de precisión y pérdida con la librería *Seaborn*. En la Figura 41 se puede observar que no parece existir sobreajuste, sin embargo, se puede observar que la precisión a pesar de mejorar, no supera el 85 %. En la Figura 42 se observa que esta precisión se ve afectada por dos clases en específico, la 2 y la 4, las cuales son la posición/gesto central y la sonrisa.

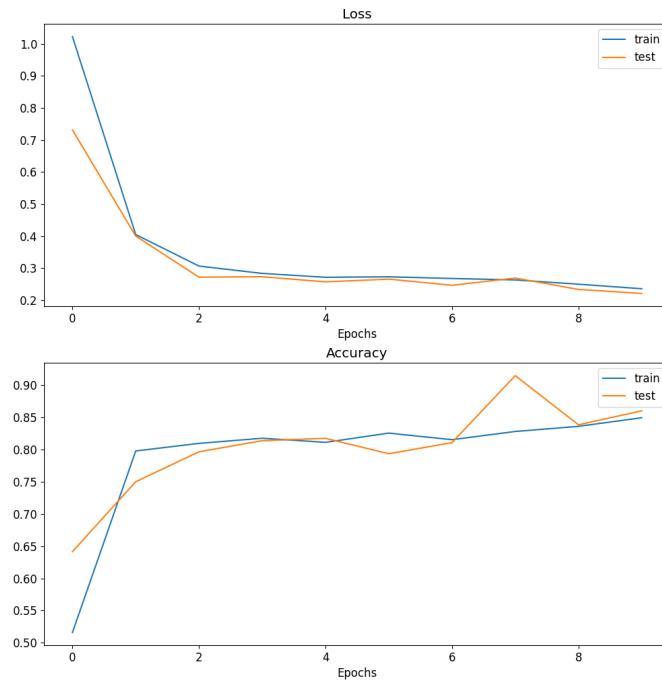


Figura 41: Pérdida y exactitud del modelo con 4 posiciones/gestos (sonrisa). (Generado por Gerardo Fuentes)

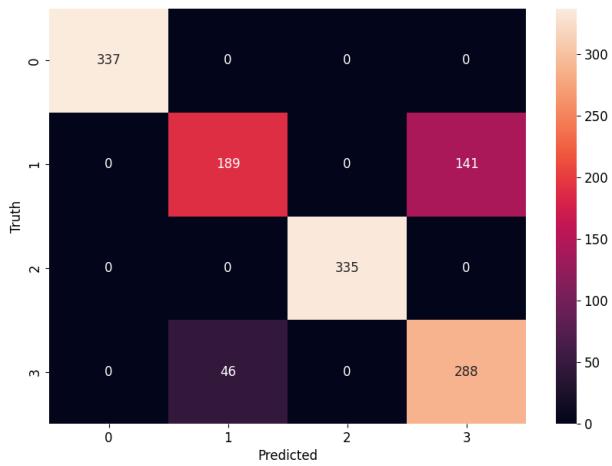


Figura 42: Matriz de confusión del modelo con 4 posiciones/gestos (sonrisa). (Generado por Gerardo Fuentes)

Para el modelo de 4 posiciones y un gesto (boca abierta) se puede observar en los resultados de las Figuras 43 y 44 son muy similares a los obtenidos en las Figuras 41 y 42. Sin embargo, se puede observar una predicción menos satisfactoria entre las clases 2 y 4, que corresponden a la posición/gesto central y la boca abierta.

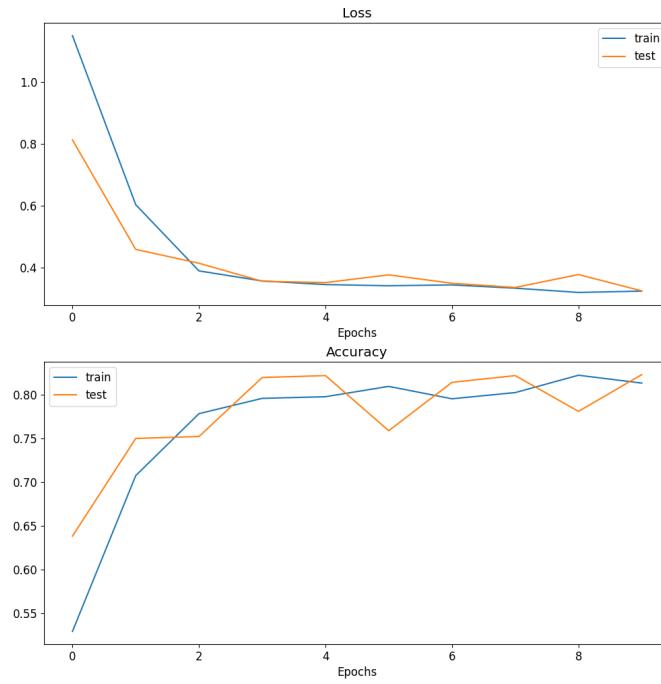


Figura 43: Pérdida y exactitud del modelo con 4 posiciones/gestos (boca abierta). (Generado por Gerardo Fuentes)

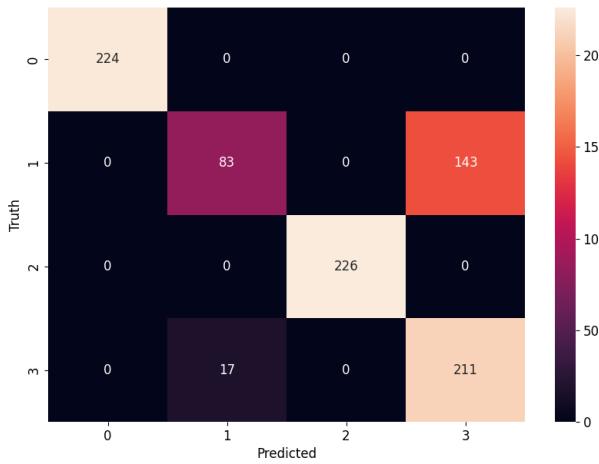


Figura 44: Matriz de confusión del modelo con 4 posiciones/gestos (boca abierta). (Generado por Gerardo Fuentes)

Ahora bien en la Figura 45, se observan resultados con 100 % de precisión, y gráficas con pérdida nula. Además esto se corrobora en la matriz de confusión de la Figura 46,

donde no se observa ninguna predicción incorrecta. En este caso la posición/gesto 4, es la posición/gesto de la cabeza hacia arriba.

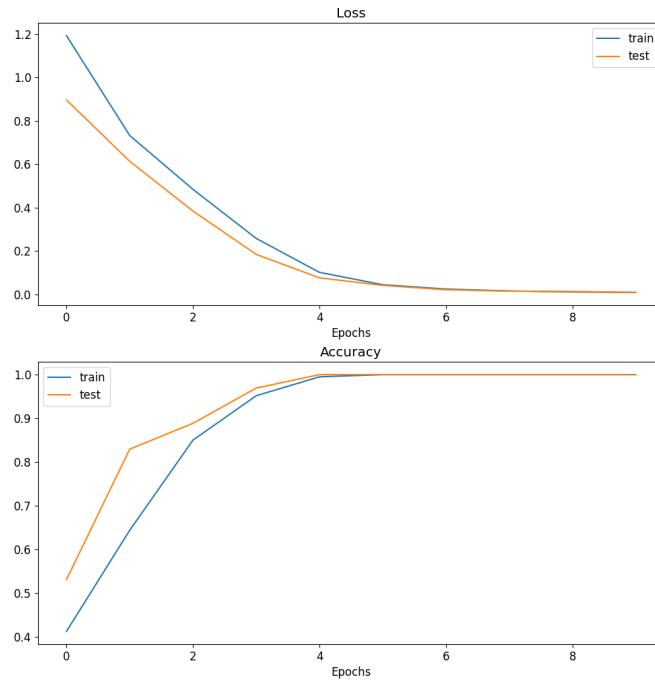


Figura 45: Pérdida y exactitud del modelo con 4 posiciones/gestos. (Generado por Gerardo Fuentes)

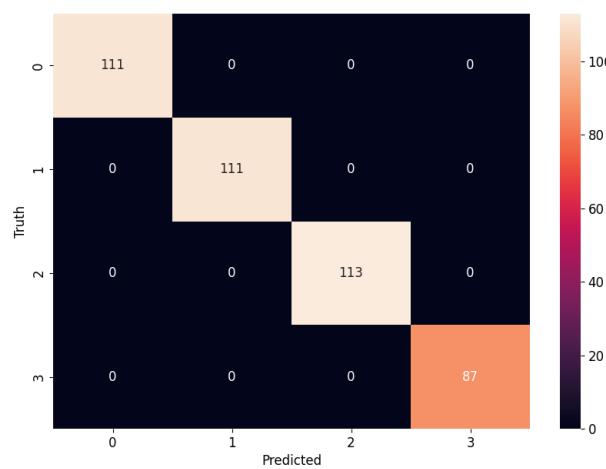


Figura 46: Matriz de confusión del modelo con 4 posiciones/gestos. (Generado por Gerardo Fuentes)

En el caso de las Figuras 47 y 48 los resultados también son prometedores. La diferencia

radica en que ahora las posiciones/gesto son: arriba, abajo, centro, derecha, izquierda.

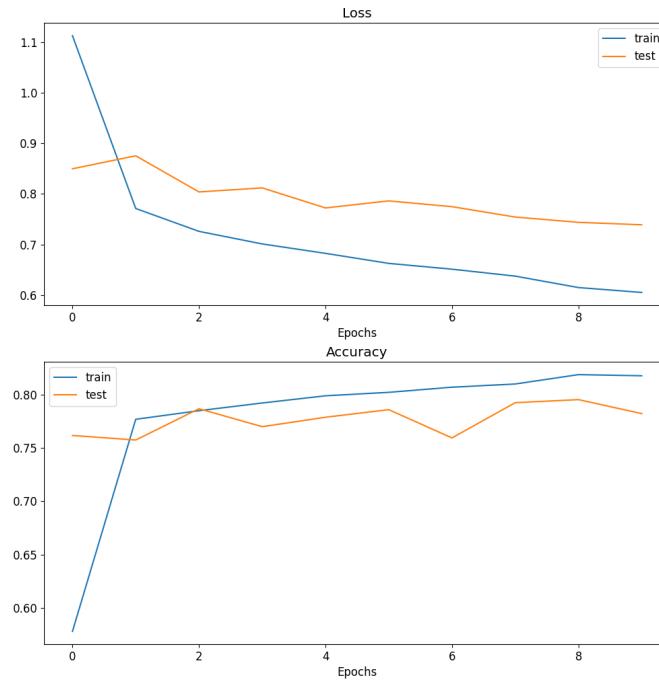


Figura 47: Pérdida y exactitud del modelo con 5 posiciones/gestos. (Generado por Gerardo Fuentes)

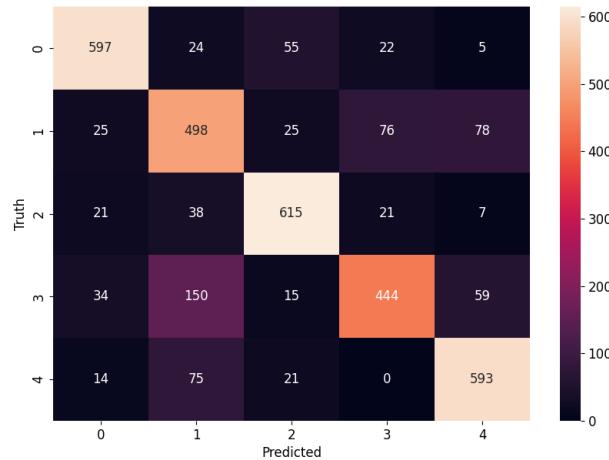


Figura 48: Matriz de confusión del modelo con 5 posiciones/gestos. (Generado por Gerardo Fuentes)

Por último, se realizó una prueba con las 9 posiciones/gestos iniciales, para verificar el rendimiento, el cual fue muy bueno, con resultados nuevamente sin errores en las Figuras

49 y 50. Sin embargo, las aplicaciones de movimiento serán afrontadas únicamente con 5 posiciones/gestos.

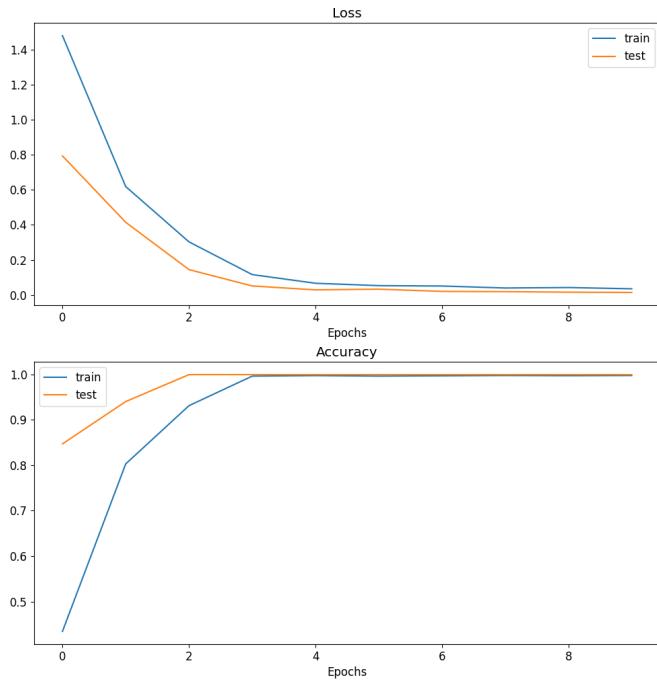


Figura 49: Pérdida y exactitud del modelo con 9 posiciones/gestos. (Generado por Gerardo Fuentes)

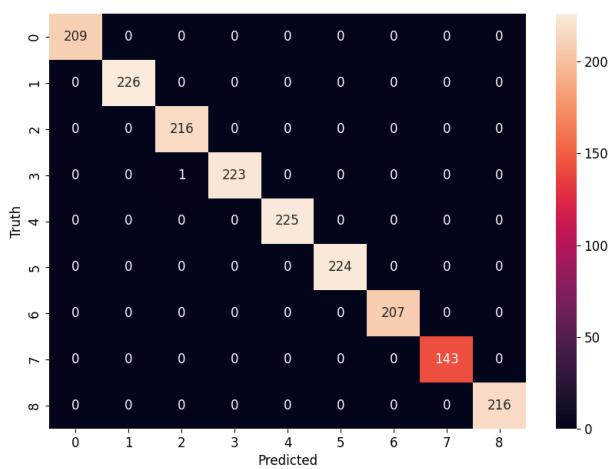


Figura 50: Matriz de confusión del modelo con 9 posiciones/gestos. (Generado por Gerardo Fuentes)

9.3. Programación multi-hilos

Dado que se trabaja con el paradigma de programación orientada a objetos, al utilizar la interfaz gráfica, junto con los módulos de simulaciones y transmisión de datos vía *Bluetooth*, sucedió que al realizarse en un mismo hilo de operaciones, es decir de manera secuencial, la interfaz no respondía y se congelaba. Por lo que se optó por utilizar la librería de *Python: Threading*. Dicha librería es capaz de colocar ciertos procesos largos en otros hilos, para poder realizar tareas de manera simultánea. Sin embargo, esto fue exitoso para los casos en que se utilizó *Turtle* y *Bluetooth*. Para el caso de la simulación en *Webots*, aún no es posible, dado que el código está diseñado para funcionar únicamente si la aplicación está abierta, por lo que únicamente se separó el proceso de encendido y apagado de la cámara.

9.4. Paro de emergencia

Dado que esta es una aplicación con el fin de controlar sistemas robóticos móviles, es necesario que exista una forma de detener el movimiento de la forma más rápida pero segura posible. En este caso, luego de obtener el trazado de puntos generado por *Mediapipe*, se calcula la diagonal entre las coordenadas del labio interno superior e inferior. El valor de dicha diagonal, por medio de pruebas con la cámara se determinó que el valor dicha diagonal que corresponde al gesto de abrir la boca, es de 0.03 % de la resolución de la cámara. Además, este paro funciona de tal manera que desacelera a una velocidad mayor que los comandos de aumento de velocidad, lo cual evita un movimiento brusco en el motor que pueda llegar a ser perjudicial para el agente robótico.

CAPÍTULO 10

Validación del sistema por medio de simulaciones

Para poder realizar las pruebas finales, es decir con plataformas robóticas físicas, es necesario comprobar que los comandos establecidos están respondiendo como se espera, más aún cuando se busca tener un proyecto funcional con capturas en tiempo real. Es por eso que, se creó una interfaz capaz de poder realizar todo el proceso virtual. Dicho proceso contempla la captura de datos, el entrenamiento con no más que una interacción con botones y luego la incorporación de simuladores que estén disponibles justo después de haber realizado el proceso de captura. A continuación se detalla la creación y modificaciones de la interfaz, así como los resultados de las simulaciones en tiempo real desde simulaciones simples utilizando la impresión de datos en la consola de *Python*, la herramienta graficadora *Turtle* de *Python* y la plataforma de simulación de robots *Webots*.

10.1. Interfaz 1(Grupos de modelos: 1 a 3)

10.1.1. Prueba en tiempo real

Para las primeras pruebas se utilizaron los 4 modelos exportados, la cámara integrada y un bucle infinito. En este caso es utilizaron las siguientes librerías *CV2*, *TensorFlow* y *Numpy*. El primer paso de la configuración fue cargar el modelo previamente exportado. Luego, se realiza la inicialización de la cámara integrada por medio de *OpenCV*, y se ajusta el tamaño de la imagen a obtener por medio de comandos de la librería *CV2*. Ahora bien, en el bucle infinito se realiza una captura de la imagen obtenida por la cámara, la cual se guarda como una matrices de píxeles. Luego, se cambia el canal de BGR a RGB, se normaliza la matriz al dividirla entre 255, y se cambia la matriz de tipo flotante a entero. Después, por medio de la librería *TensorFlow* se utiliza la función Predecir, la cual devuelve una vector con el porcentaje de similitud predecido para cada clase. Finalmente, se aplica la función de *Numpy* para obtener el valor más máximo dentro de dicho vector de predicciones y dicho valor se imprime en la consola inicialmente.

10.1.2. Bloques de funciones

La primera versión de la interfaz creada, utiliza 3 grandes grupos de códigos. El primer grupo se encarga de capturar fotos y clasificarlas según la etiqueta que se necesite utilizar, en este caso se trabaja con 4. En esta ventada se presentan instrucciones en forma de lista para que el usuario pueda identificar y capturar las fotografías, por medio de una cuadrícula de selección. Luego, el segundo bloque cuenta con las opciones necesarias para preparar las imágenes de tal manera que puedan ser interpretadas por los modelos de programación. Por último, está el bloque de prueba en vivo, esta utiliza el código descrito en la sección “Prueba en tiempo real”. Además, es importante resaltar que todos estos bloques funcionan con códigos los códigos utilizando en el capítulo “Algoritmos” dado que estos fueron construidos con programación orientada a objetos. Por último, para la creación de esta interfaz se utilizaron las librerías *Tkinter* y *Custom Tkinter* respectivamente. Las pestañas para la utilización de la interfaz se presentan en las Figuras 51, 52 y 53.

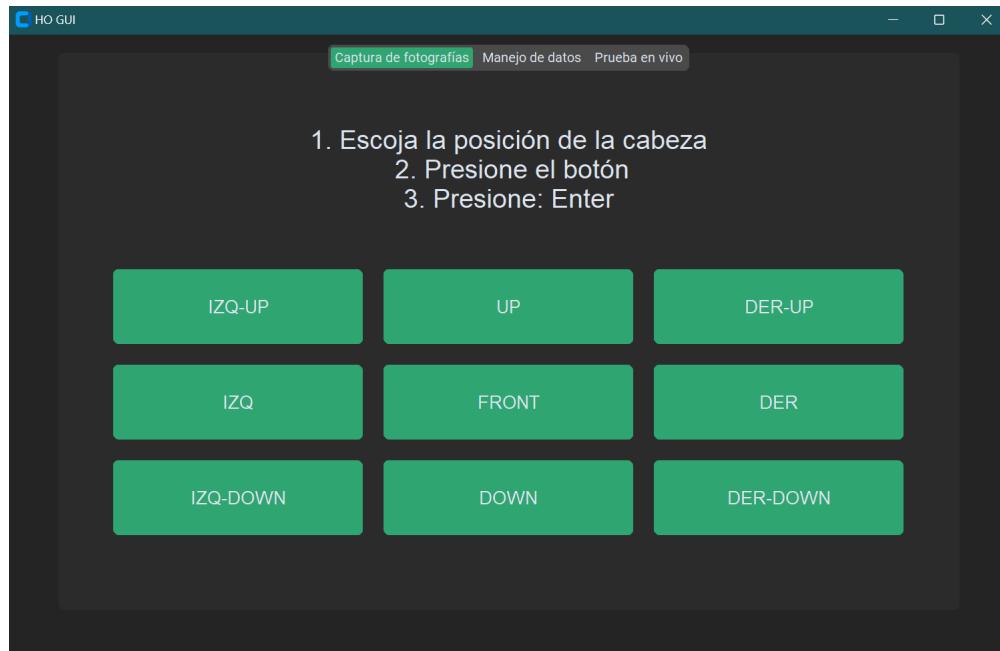


Figura 51: Sección de captura de fotos en la interfaz. (Generado por Gerardo Fuentes)

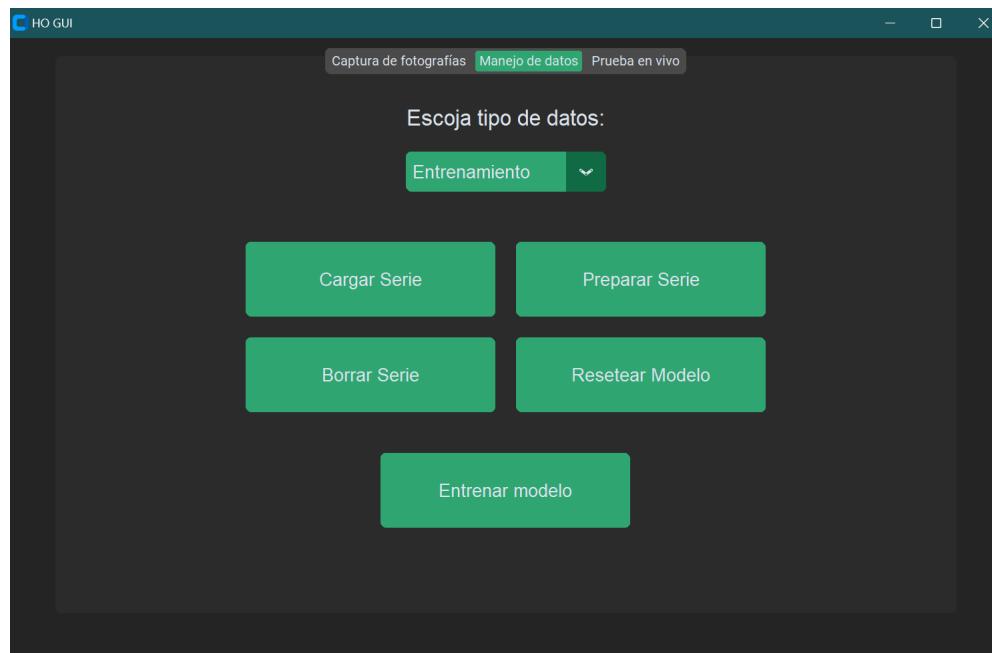


Figura 52: Sección de manejo de datos en la interfaz. (Generado por Gerardo Fuentes)

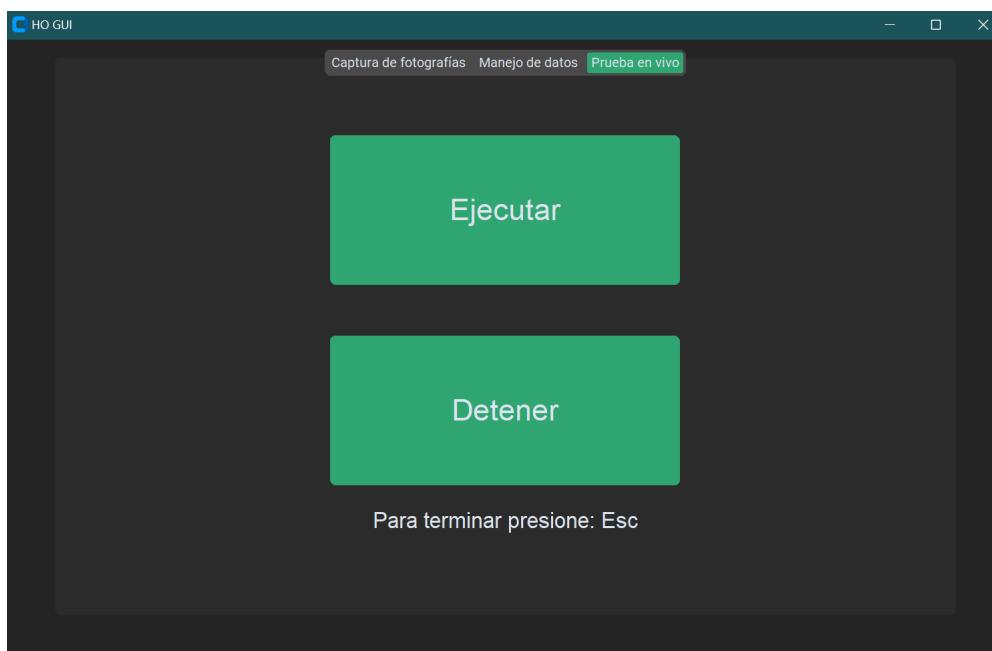


Figura 53: Sección de pruebas en tiempo real con *Turtle*. (Generado por Gerardo Fuentes)

10.2. Interfaz 2 (Grupo de modelos: 4)

Luego de realizar cambios respecto a los modelos de aprendizaje, también se optimizó la cantidad de operaciones que tiene que realizar el usuario para utilizar la interfaz. Esto

fue posible, por medio de la combinación de funciones pequeñas en una sola más grande. Así también, se optimizaron los tiempos de captura de datos como se describe y se sigue utilizando el paradigma de programación orientado a objetos, los resultados se muestran a continuación:

10.2.1. Captura de datos

Previamente se contaba con una pestaña de captura de datos, en la cual al presionar un botón, se capturaba una sola posición/gesto, por lo que era muy difícil obtener una gran cantidad de datos, teniendo en total 9 botones. En esta nueva versión, es posible hacer, la captura de hasta 300 fotografías por posición, al presionar un solo botón y siguiendo instrucciones que se muestran en pantalla. El cambio se percibe en el espacio libre como lo muestra la Figura 54

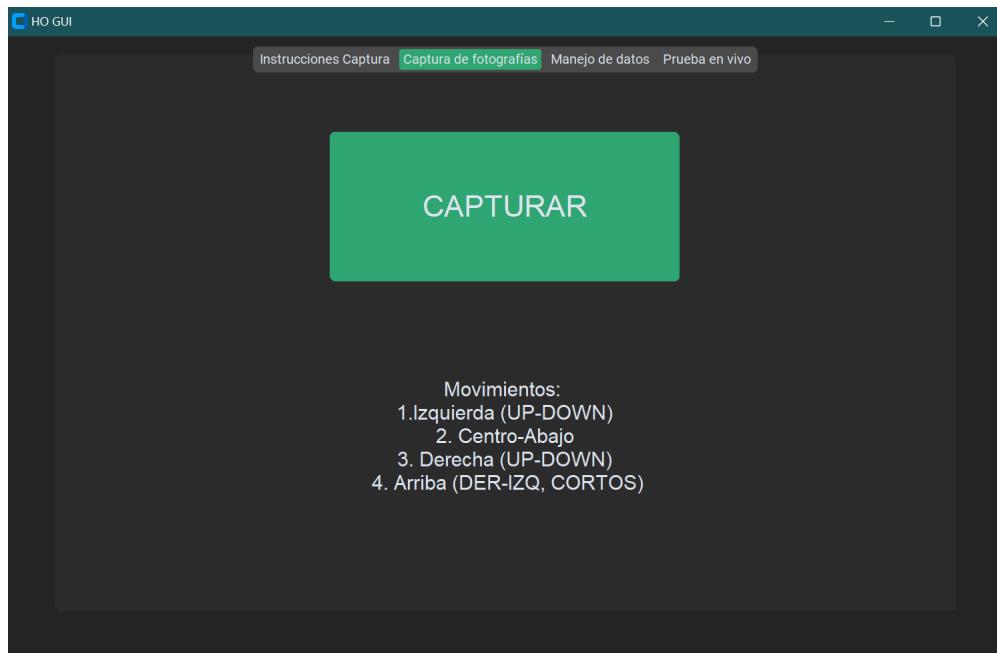


Figura 54: Sección de captura de fotos en la interfaz, versión 2. (Generado por Gerardo Fuentes)

10.2.2. Manejo de datos

Anteriormente, el usuario tenía la tarea de clasificar, las fotografías capturadas como entrenamiento o validación. Luego tenía que presionar un botón para cargar, otro para preparar las imágenes y uno último para entrenar el modelo. Ahora es posible realizar estas 3 actividades con un solo botón. Sin embargo, los botones para borrar la series de datos actual y para restablecer los datos del modelo siguen existiendo, dado que son necesarios. La nueva disposición de botones se presenta en la Figura 55

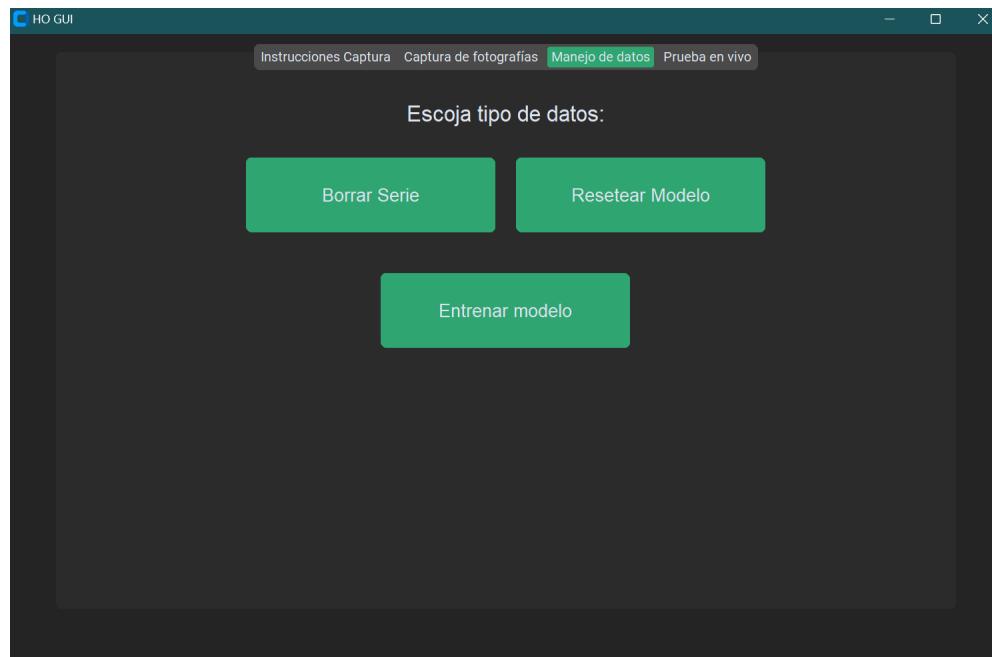


Figura 55: Sección de manejo de datos en la interfaz, versión 2. (Generado por Gerardo Fuentes)

10.3. Interfaz 3

Dado que ya se cuenta con un modelo de reconocimiento facial confiable, se procedió a agregar dos nuevas pestañas a la interfaz gráfica, de tal manera que ahora se pueden realizar pruebas *Turtle*, *Webots* y la verificación de recepción de señales físicas por parte del ESP32, por separado con sus respectivos botones de inicio y apagado, con la facilidad de cambiar de uno a otros sin problemas de congelamiento en la pantalla. Sin embargo, en esta fase, se requiere que el archivo de *Webots* que contiene el mundo creado para la simulación, se abra y se mantenga abierto durante la utilización de la interfaz gráfica. Las nuevas pestañas para manejo de simulaciones y envío de datos se observan en la Figura 56

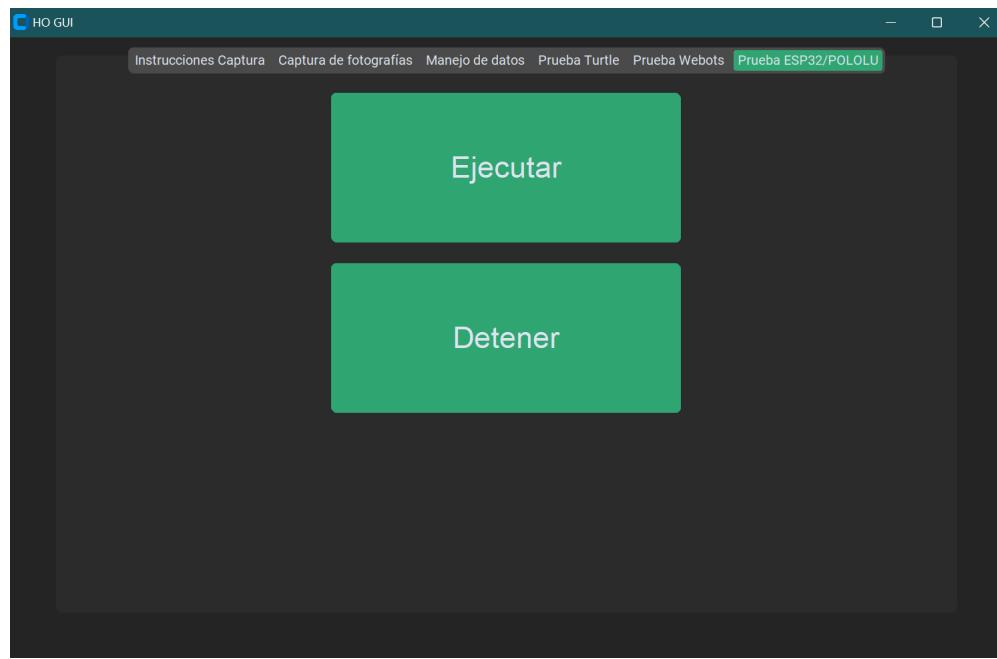


Figura 56: Interfaz gráfica con 3 métodos de traducción de predicción a comandos. (Generado por Gerardo Fuentes)

10.3.1. Captura de datos con apoyo audiovisual

Para hacer de manera más intuitiva la captura de datos para un usuario nuevo, se decidió agregar una voz artificial genérica generada por medio de una página web a partir de texto. Los textos fueron redactados de la manera más amigable posible, para tener un audio claro y agradable a los usuarios. Además, estos audios fueron acompañados por una serie de figuras que se mostraban en pantalla e indicaban de manera visual la posición/gesto, que el usuario debe realizar. A continuación, en la Figura 57 se puede observar la referencia de imágenes utilizada para dicha ayuda visual:

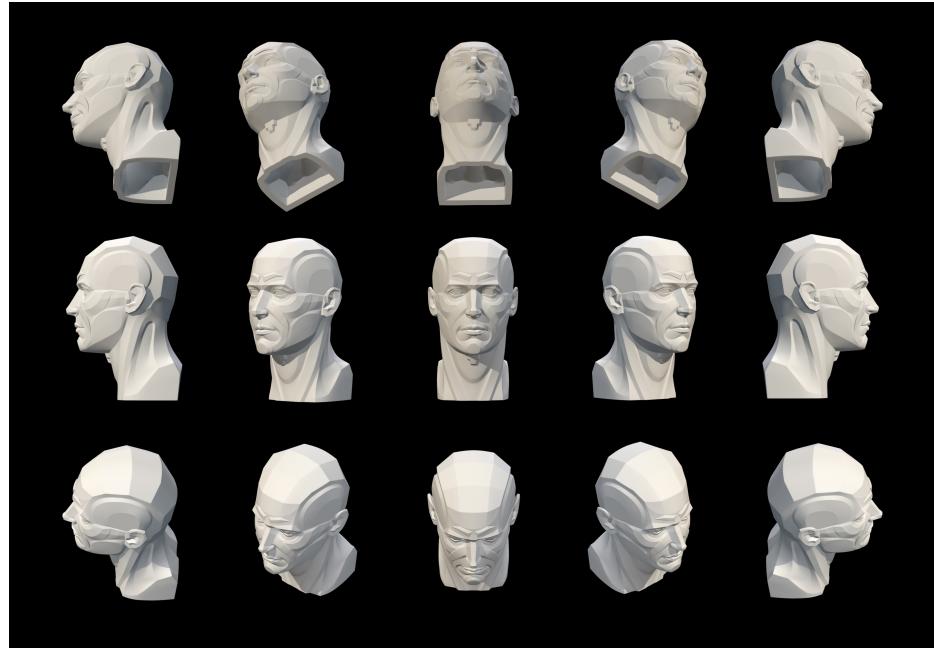


Figura 57: Posiciones de referencia para apoyo a usuarios. [42]

10.4. Consola (Grupo de modelos: 1)

En el caso de los primeros modelos realizados, la experiencia fue pésima, dado que el modelo predecía en su mayoría las posiciones inferiores, tanto para los modelos con 9 y 6 clases. Se intentó realizar dicho proceso de movimiento a diferentes distancias de la cámara pero el resultado fue el mismo. Para visualizar los resultados de estos modelos, se utilizó el código de prueba en tiempo real, imprimiendo valor de la predicción del modelo en la consola, como estaba previsto inicialmente

10.5. *Turtle*

10.5.1. Grupo de modelos: 2

En cuanto a los modelos de 4 y 3 clases, el resultado depende de la distancia de separación del rostro del usuario y de la cámara. A más de 30 cm de distancia, las predicciones son bastante malas, dado que muchas veces se la predicción es errónea en cuanto al lado de la orientación de la cabeza, lo que generaba giros no intencionales. Sin embargo, al realizar pruebas justo a 30 cm de la cámara, tanto para el modelo de 3 y 4 clases, el resultado fue satisfactorio, pero incómodo para el usuario. Esto puede indicar que falta entrenar al modelo con diferentes tipos de fondos, dado que cuando la cámara capta en su mayoría el rostro de la persona, las predicciones son buenas. Además, se realizó la prueba a más de 30 cm en diferentes espacios de trabajo en la casa de Gerardo Fuentes, y los resultados no fueron buenos, hasta que se realizó el acercamiento nuevamente. En este caso, ya se contaba con

la primera versión de la interfaz que es capaz de capturar y manipular imágenes, así como graficar por medio de la librería *Turtle* de *Python* el movimiento de un cursor, así como su trayectoria. En la Figura 58, se muestra la trayectoria realizada al intentar dibujar un cuadro.

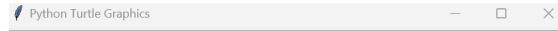


Figura 58: Prueba con trayectoria de un cuadrado en *Turtle* (modelos 2). (Generado por Gerardo Fuentes)

10.5.2. Grupo de modelos: 4

Con este grupo de modelos, las distancias ya no fueron medidas, dado que el funcionamiento tienen un rango mucho mayor. La detección de posiciones/gestos es sumamente efectiva. Además, ahora se cuenta con 5 posiciones, por lo que la función de reversa fue puesta en práctica y realizada satisfactoriamente. Por último, se cambiaron las funciones de movimiento, ahora una posición/gesto hacia arriba acelera adelante y hacia abajo genera la reversa, mientras que girar la cabeza a los lados proporciona el cambio de dirección en movimiento y sobre su misma posición si no hay aceleración. A continuación, la Figura 59, deja ver un cuadro bastante claro.

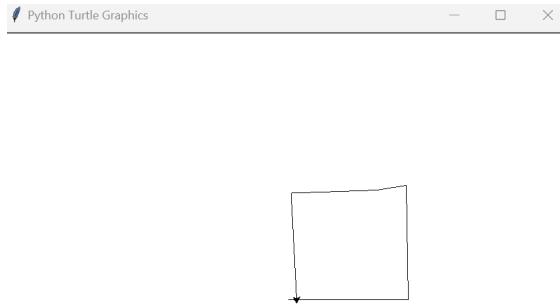


Figura 59: Prueba con trayectoria de un cuadrado en *Turtle* (modelos 4). (Generado por Gerardo Fuentes)

10.6. *Webots*

10.6.1. Grupo de modelos: 3

En esta simulación fue necesario utilizar el software *Webots*, crear un mundo simple, que contuviera un piso rectangular y el robot a utilizar. En este caso se optó por el modelo del robot *E-puck*, dado que es un agente robótico móvil de dos ruedas, muy similar a una silla de ruedas. Además, se configuró el robot para utilizar un controlador externo, en vez de la opción por defecto, que permite utilizar un archivo incluido en la interfaz del software. La selección de un controlador externo fue realizada, dado que se requiere utilizar tanto las librerías del robot, como las que se han implementado para el manejo de datos y la predicción de clases en tiempo real. También es importante resaltar, que el código utilizado para manejar la simulación en *Webots*, aún no se ha incorporado a la interfaz.

En estas simulaciones, el séptimo y octavo modelos funcionaron de manera similar. En esta ocasión, la predicción de las posiciones funcionaba en un rango mayor de distancias, sin embargo existía una tendencia a predecir posiciones centrales de la cabeza como rotaciones a la derecha. Sin embargo, este último problema parecía no aparecer cuando el usuario se posicionaba con un desfase respecto de la cámara hacia la izquierda. Por último, el noveno modelo funciona en un mayor rango de posiciones, distancias y fondos, aunque requiere utilizar los marcadores faciales, incluso así, llegando a presentar los problemas mencionados con el séptimo y octavo modelos, pero con menor error. Además, estos resultados fueron obtenidos al ser Gerardo Fuentes (autor) el usuario, dado que con usuarios nuevos, el resultado aún no es óptimo. Todo esto se consiguió, con una exactitud menor al 70 % que se observó en las gráficas de dicho modelo en la sección de “ Tercer grupo de modelos de aprendizaje” en el capítulo “Algoritmos”. A continuación, se observa la trayectoria recorrida por el robot simulado cuando se intentó realizar un cuadro, en la Figura 60.

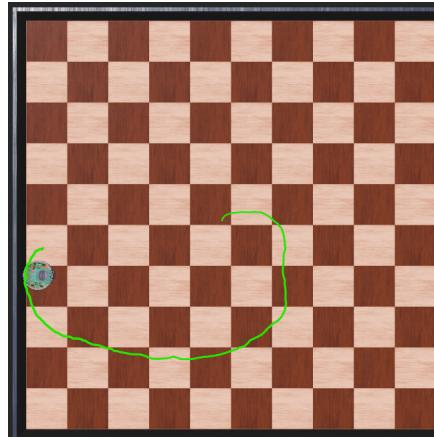


Figura 60: Prueba con trayectoria de un cuadrado en *Webots* (modelos 3). (Generado por Gerardo Fuentes)

10.6.2. Grupo de modelos: 4

En el caso de la simulación en *Webots*, aún se realizaron pruebas sin la incorporación con la interfaz, sin embargo los resultados de movimiento fueron igual de satisfactorios que los realizados con *Turtle*. Sin embargo, aún queda ajustar los valores de las velocidades a utilizar. En la Figura 61, se puede observar la trayectoria realizada.

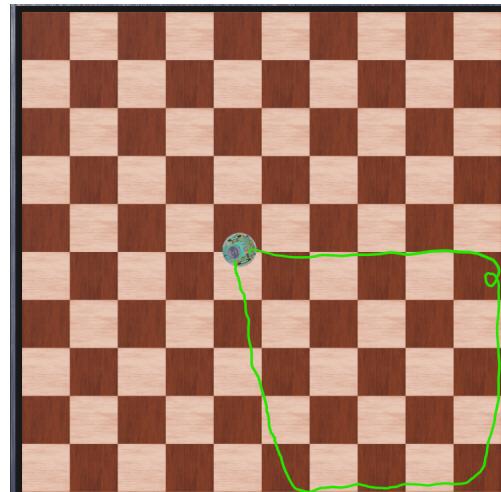


Figura 61: Prueba con trayectoria de un cuadrado en *Webots* (modelos 4). (Generado por Gerardo Fuentes)

CAPÍTULO 11

Validación del sistema con plataformas robóticas físicas

11.1. Verificación de recepción de señales

11.1.1. PC-PYTHON

El enfoque principal de estas pruebas es la utilización de un protocolo de transmisión de datos que permita una alta confiabilidad tanto en la conexión como una eficiencia de espacio proyectada a futuro. Estas condiciones fueron propuestas dado que se espera que el agente robótico pueda responder a los comandos por medio de una conexión inalámbrica. Teniendo en cuenta estas condiciones se optó por utilizar el estándar de comunicación *Bluetooth*, para lo cual se intentó realizar una comunicación iniciada por un código de python que implementa la librería *Socket*, con la cual solo hay especificar la dirección física del dispositivo cliente (*ESP32*), un canal de comunicación, iniciar el controlador del *Bluetooth* de la *PC* y realizar la codificación del dato a enviar. En este caso el dato surge a partir de la utilización del modelo de aprendizaje previamente guardado. De esta manera, se elimina la dependencia de un emparejamiento desde la configuración de *Windows*. Sin embargo, en las primeras pruebas estos no fue posible. La alternativa fue realizar un emparejamiento del *ESP32* junto con la *PC*, desde la configuración de *Windows*, dicho proceso solo implica habilitar los módulos de ambos dispositivos y emparejar como cualquier otro dispositivo que cuente con esta tecnología. Por último, al estar emparejada la placa *ESP32*, esta ocupa un puerto de comunicación serial, por lo que la programación fue realizada utilizando los conceptos de la comunicación serial. Se estableció el número de puerto, el baudaje, y se envió el valor de la predicción del modelo en codificación del tipo *Char*.

11.1.2. ESP32/PROTOBOARD

En el caso de la placa *ESP32*, se realizó una configuración de comunicación serial sencilla, asegurando que el puerto y el baudaje sea el mismo, así como una decodificación del mensaje recibido en tipo *Char*. Sin embargo, a diferencia de las simulaciones, dónde el código de

Python contenía el algoritmo de traducción de predicción a comandos, en este caso la placa *ESP32* tuvo dicha función. Dado que esta etapa es de verificación, se configuraron 4 diodos emisores de luz que responden de diferente manera dependiendo de la predicción. En este caso, la posición de la cabeza arriba: enciende el diodo que emite luz verde que se encuentra junto al diodo de luz roja. La posición de cabeza abajo: enciende el diodo que emite luz verde que se encuentra junto al diodo de luz azul. La posición de la cabeza derecha: enciende el diodo de luz roja, y la posición de cabeza izquierda: enciende el diodo de luz azul. La disposición de este sistema de visualización de señales se puede observar en la Figura 62 Por último, la posición de cabeza central: apaga todos los diodos. En este caso, la prueba fue un éxito y está lista para realizar los ajustes necesarios para los comandos de movimiento para el agente robótico móvil: Pololu.

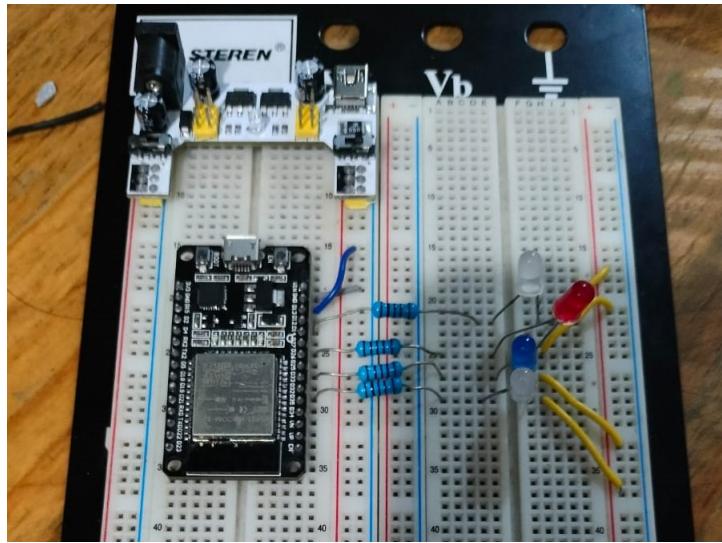


Figura 62: Configuración de pruebas para verificación de recepción de predicciones. (Generado por Gerardo Fuentes)

11.2. Verificación de funcionamiento con agente robótico móvil

Para poner a prueba el funcionamiento completo del sistema, se utilizaron los agentes robóticos Pololu 3pi+ que se encuentran disponibles en la Universidad del Valle de Guatemala. Se determinó que se utilizaría este agente dado que se han realizado pruebas exitosas con muchos proyectos del departamento, con diferentes tipos de comunicación. En este caso, se aprovechó la placa desarrollada en el departamento de Ingeniería Electrónica, Mecatrónica y Biomédica de la Universidad del Valle de Guatemala, con la cual se puede conectar los cuatro pines necesarios para comunicación serial desde el *ESP32 DEVKIT C V4* hacia el Pololu, de una manera físicamente estable. Luego se programó el microcontrolador *ESP32 DEVKIT C V4* en el entorno de desarrollo integrado (IDE) de Arduino, aquí se crearon comandos de *FreeRTOS*, uno para recibir los datos de la cámara por medio de *Bluetooth* serial y generar controlar las velocidades de las llantas y otro para serializar los datos en formato binario (CBOR específicamente) de dichas velocidades y enviarlas al Pololu. Dicha

placa se puede observar en diferentes perspectivas por medio de la Figura 63 y 64:

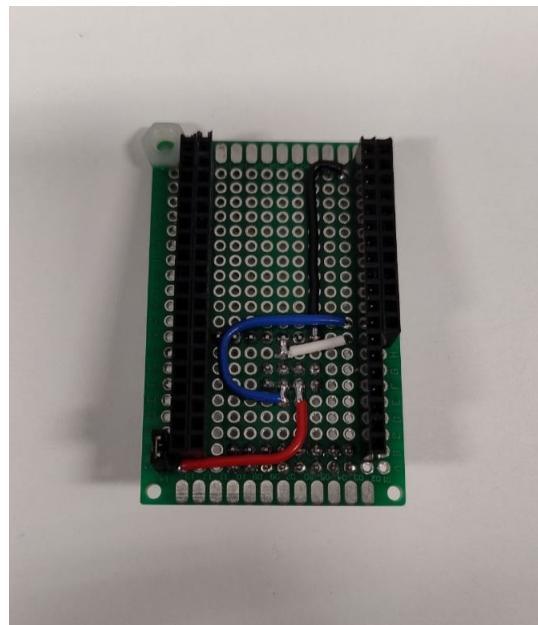


Figura 63: Placa para comunicación serial *ESP32-Pololu* (vista superior). (Generada por Gerardo Fuentes)

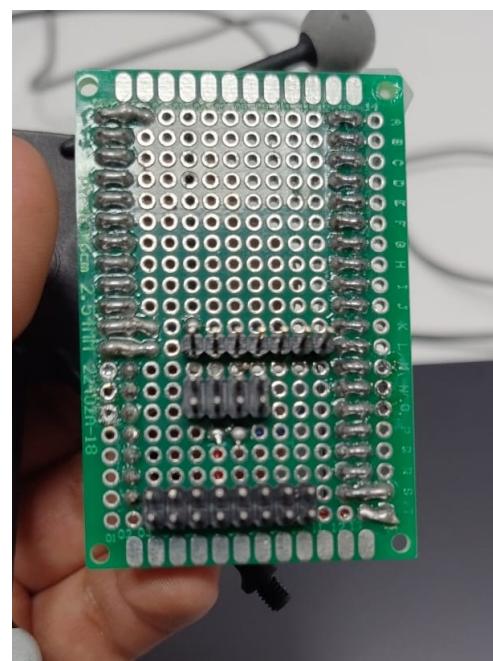


Figura 64: Placa para comunicación serial *ESP32-Pololu* (vista inferior). (Generada por Gerardo Fuentes)

11.2.1. Antirrebotes

Al realizar las pruebas con el agente robótico físico, se observó un comportamiento erróneo al predecir la posición del usuario y otro poco intuitivo con el manejo de velocidades. Al verificar los comandos y las señales enviadas por medio de *Bluetooth* serial, se observó que el comportamiento erróneo sucede al momento de realizar una predicción al cambiar de la posición/gesto central de la cabeza hacia los lados. Sin embargo, este error consiste en una predicción errónea únicamente al hacer el gesto, por lo que dura muy poco pero es suficiente como para realizar una acción no esperada. Ahora bien con las velocidades, resulta poco o nada práctico aumentar la velocidad en un valor constante cada vez que se eleva la cabeza o se posiciona hacia abajo. Luego de analizar estas situaciones, se decidió hacer dos tipos de antirrebote. El primero funciona de tal manera que si una predicción se realiza una única vez y luego cambia a otra, se considera error y no se realiza la acción correspondiente, sin embargo, este primer antirrebote podría ser innecesario si al alimentar el modelo de predicción con más datos, el error se mitiga. El segundo antirrebote, para la velocidad, consta de incrementar el valor de la velocidad en un porcentaje fijo si se realiza el gesto respectivo, pero si la posición persiste por más de cierta cantidad de ciclos, esta incrementará de manera continua hasta llegar al límite de las velocidades del Pololu. En la Figura 65, se puede observar el posicionamiento del usuario (Gerardo Fuentes), para utilizar el sistema desarrollado.



Figura 65: Utilización del sistema de reconocimiento de gestos y posiciones para controlar el agente robótico móvil. (Generada por Gerardo Fuentes)

CAPÍTULO 12

Conclusiones

- La selección de hardware resultó útil, dado que todas las pruebas fueron realizadas con la cámara de la laptop, y la verificación física únicamente requirió una *Protoboard*, cables, diodos emisores de luz, una fuente de alimentación y la placa *ESP32*.
- La selección del software fue útil durante el proyecto, dado que se pudo experimentar con diferentes tipos de librerías para completar las funciones requeridas por el sistema y plataformas como *Mediapipe* prometen seguir en el mercado por varios años.
- Se validó que los algoritmos de visión por computadora no tuvieron resultados exitosos cuando se combinaron métodos de reconocimiento clásicos, como filtros o marcadores físicos en las imágenes de las bases de datos para alimentar los modelos de aprendizaje de máquina.
- El algoritmo de visión de computadora tuvo resultados exitosos gracias a la combinación de la herramienta *Mediapipe*, la cual permitió obtener los puntos de la cara, la cual se puede interpretar como un grafo y la implementación de la red neuronal de grafos.
- El algoritmo de traducción de gestos fue creado pensando en el movimiento de un automóvil automático, lo cual funcionó de manera satisfactoria, al utilizar los gestos de movimiento de cara hacia arriba y abajo para avanzar y retroceder, así como controlar la dirección con el movimiento hacia izquierda o derecha.
- La validación por medio de simulaciones pudo implementarse de manera correcta al programar el graficador *Turtle* y al configurar de manera exitosa la conexión entre *Webots* y el entorno de programación *PyCharm Community Edition* como un controlador externo.
- La validación por medio de comunicación Bluetooth, permitió determinar que el envío de datos fuera correcto utilizando la cámara de la laptop y una placa *ESP32* que se encarga de manejar los comandos a utilizar en la plataforma robótica.

- El costo computacional de utilizar el lenguaje de programación *Python*, se ve reflejado cuando no se programa adecuadamente el hilado de procesos, ocasionando congelamiento de funciones.
- El costo computacional de trabajar con *Python* se ve compensado con la forma sencilla e intuitiva para configurar redes neuronales de diferentes tipos como convolucionales y de grafos.
- Las simulaciones realizadas desde la plataforma *Webots*, son valiosas, sin embargo, para un modelo predictivo la simulación con *Turtle* genera una representación útil para la verificar el funcionamiento del modelo.
- Usar programación orientada a objetos permitió implementar una interfaz gráfica la cual es fácilmente depurable.

CAPÍTULO 13

Recomendaciones

- Dado que el código fue desarrollado en Python, tiene mucho peso computacional a comparación de otros lenguajes de programación, sería un reto importante poder hacer una migración de este sistema a lenguajes como C, dado que es superior en cuanto a administración de recursos computacionales.
- Dado que en la Universidad del Valle de Guatemala se cuenta con el sistema *Opti-Track*, y muchas aplicaciones de laboratorios de Robótica se realizan en el ecosistema MATLAB, también sería valioso trasladar este sistema a dicho ecosistema, para aprovechar la gran cantidad de herramientas adicionales.
- Para obtener un mejor registro de las observaciones de las pruebas realizadas, se recomienda crear una bitácora en la que se detalle el modelo o algoritmo utilizado, con cantidad de repeticiones y las observaciones realizadas para cada repetición.
- Explorar nuevos enfoques de modelos de aprendizaje sería valioso para la investigación, dado que existen muchos más tipos de redes neuronales, en este caso solo se exploraron las funciones clasificativas de las GNN y CNN, sería interesante evaluar el potencial con los modelos generativos. De esta manera verificar si existe una relación entre otros tipos de redes neuronales y la comodidad del usuario.
- La interfaz actual, contiene una optimización considerable de tiempos de captura de datos y procesado simultáneo de tareas. Sin embargo, siempre hay espacio para la mejora, como el caso de tener que mantener el archivo de *Webots* abierto.
- Siguiendo la línea de la interfaz, sería valioso que al lograr un eficiente manejo de los procesos, esta fuera exportada a una aplicación lista para instalación en diferentes sistemas operativos y así estar al alcance de usuarios particulares.
- Si bien el modelo de aprendizaje, funciona bien en la mayoría de los casos, sería valioso combinar nuevos modelos de aprendizaje para solventar el problema de no reconocimiento cuando la iluminación trasera es mayor que la delantera, respecto al usuario.

- Como enfoque adicional, si se logra cambiar el sistema inalámbrico *Bluetooth* por aplicaciones con *Wi-Fi* y *IoT*, se podría explotar el potencial explorativo que se tiene, como asistente en situaciones de ayuda en desastres naturales de manera remota, por ejemplo.
- Implementar un sistema de seguridad contra obstáculos que funcione de manera automática, en situaciones en que el usuario sea incapaz de reaccionar.
- Dado que el modelo de aprendizaje es capaz de detectar muchas más clases, sería valioso trasladar los comandos utilizados en el agente robótico Pololu a algún tipo de dron.

CAPÍTULO 14

Bibliografía

- [1] R. Mehta y V. Jain, “Computer Vision Based Prototype Model of Face Gesture Controlled Vehicle,” jul. de 2021, págs. 313-317. DOI: 10.1109/CCICT53244.2021.00065.
- [2] S.-W. Lee, *Automatic Gesture Recognition for Intelligent Human-Robot Interaction ...* Mayo de 2006. dirección: <https://ieeexplore.ieee.org/document/1613091>.
- [3] S. Sharma, S. Jain y Khushboo, “A Static Hand Gesture and Face Recognition System for Blind People,” en *2019 6th International Conference on Signal Processing and Integrated Networks (SPIN)*, 2019, págs. 534-539. DOI: 10.1109/SPIN.2019.8711706.
- [4] R. T. Bankar y S. S. Salankar, “Head Gesture Recognition System Using Gesture Cam,” en *2015 Fifth International Conference on Communication Systems and Network Technologies*, 2015, págs. 535-538. DOI: 10.1109/CSNT.2015.81.
- [5] L. A. Rivera, G. N. DeSouza y L. D. Franklin, “Control of a Wheelchair Using an Adaptive K-Means Clustering of Head Poses,” en *2013 IEEE Symposium on Computational Intelligence in Rehabilitation and Assistive Technologies (CIRAT)*, 2013, págs. 24-31. DOI: 10.1109/CIRAT.2013.6613819.
- [6] Dirección: <https://developers.google.com/mediapipe>.
- [7] C. M. Bishop, *Pattern recognition and machine learning*. MTM, 2023.
- [8] L. Breiman, J. H. Friedman, R. A. Olshen y C. J. Stone, *Classification and regression trees*. CRC Press, 2017.
- [9] I. Goodfellow, Y. Bengio y A. Courville, *Deep learning*. MITP, 2018.
- [10] C. C. Aggarwal, *Neural networks and deep learning*. dirección: <https://link.springer.com/book/10.1007/978-3-319-94463-0>.
- [11] A. Williams, *Convolutional neural networks in Python: Introduction to convolutional neural networks*. Amazon Fulfillment Poland Sp. z o.o, 2017.
- [12] MATLAB, ¿Qué son las redes neuronales convolucionales?: 3 cosas que debe saber. dirección: <https://es.mathworks.com/discovery/convolutional-neural-network.html>.
- [13] W. L. Hamilton, *Graph representation learning*. Morgan y Claypool Publishers, 2020.

- [14] B. Gtz, *Introduction to graph neural networks: An illustrated guide*, mayo de 2023. dirección: <https://medium.com/@bscarleth.gtz/introduction-to-graph-neural-networks-an-illustrated-guide-c3f19da2ba39>.
- [15] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer, 2023.
- [16] IBM, *What is Computer Vision?* n.d. dirección: <https://www.ibm.com/topics/computer-vision>.
- [17] A. Krizhevsky, I. Sutskever y G. E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*, n.d. dirección: https://www.researchgate.net/publication/267960550_ImageNet_Classification_with_Deep_Convolutional_Neural_Networks.
- [18] MathWorks, n.d. dirección: <https://www.mathworks.com/discovery/object-detection.html>.
- [19] V. Lepetit, F. Moreno-Noguer y P. Fua, “EPnP: An Accurate O(n) Solution to the PnP Problem,” vol. 81, feb. de 2009. DOI: 10.1007/s11263-008-0152-6.
- [20] M. Walia, *What Is Object Tracking in Computer Vision?* Dic. de 2022. dirección: <https://blog.roboflow.com/what-is-object-tracking-computer-vision/>.
- [21] A. Acharya, *The Complete Guide to Object Tracking [Tutorial]*, oct. de 2023. dirección: <https://encord.com/blog/object-tracking-guide/>.
- [22] N. Barla, *The Complete Guide to Object Tracking [+V7 Tutorial]*, nov. de 2021. dirección: <https://www.v7labs.com/blog/object-tracking-guide>.
- [23] N. Klingler, *Object Tracking in Computer Vision (2024 Guide)*, nov. de 2023. dirección: <https://viso.ai/deep-learning/object-tracking/>.
- [24] X. Wang, J. Jang, Y. Wei, L. Kang e Y. Gao, *Research on Gesture Recognition Method Based on Computer Vision Technology*, nov. de 2018. dirección: https://www.matec-conferences.org/articles/matecconf/abs/2018/91/matecconf_eitce2018_03042/matecconf_eitce2018_03042.html.
- [25] M. RadhaKrishna, *A Review on Image Processing Sensor*, 2021. dirección: <https://iopscience.iop.org/article/10.1088/1742-6596/1714/1/012055>.
- [26] X. Wang, *The Evolution of LIDAR and Its Application in High Precision ...* n.d. dirección: <https://iopscience.iop.org/article/10.1088/1755-1315/502/1/012008>.
- [27] C. Atwell, *What Is a Radar Sensor?* Mar. de 2021. dirección: <https://www.fierceelectronics.com/sensors/what-a-radar-sensor>.
- [28] R. Ingle, *How Do Radar Sensors Work?* Ago. de 2023. dirección: <https://www.azosensors.com/article.aspx?ArticleID=2854>.
- [29] L. Li, *Time-of-Flight Camera – An Introduction*, 2014. dirección: <https://www.ti.com/lit/wp/sloa190b/sloa190b.pdf>.
- [30] M. Andersen, T. Jensen, P. Lisouski et al., “Kinect Depth Sensor Evaluation for Computer Vision Applications,” *Technical Report Electronics and Computer Engineering*, vol. 1, n.º 6, sep. de 2012. dirección: <https://tidsskrift.dk/ece/article/view/21221>.
- [31] N. Mahamkali y V. Ayyasamy, “OpenCV for Computer Vision Applications,” mar. de 2015.

- [32] M. Abadi, P. Barham, J. Chen et al., “TensorFlow: A System for Large-Scale Machine Learning,” mayo de 2016.
- [33] D. Kirk, “NVIDIA CUDA Software and GPU Parallel Computing Architecture,” vol. 7, oct. de 2007, págs. 103-104. DOI: 10.1145/1296907.1296909.
- [34] MathWorks, n.d. dirección: <https://www.mathworks.com/discovery/computer-vision.html>.
- [35] Berkeley AI Research, n.d. dirección: <https://caffe.berkeleyvision.org/>.
- [36] M. D. Seyer, *RS-232 made easy: Connecting computers, printers, terminals and modems*. Prentice Hill, 1991.
- [37] A. S. Huang y L. Rudolph, *Bluetooth essentials for programmers*. Cambridge University Press, 2007.
- [38] L. J. Zhi, *Sistemas operativos en tiempo real (RTOS) y Sus Aplicaciones*, feb. de 2021. dirección: <https://www.digikey.com/es/articles/real-time-operating-systems-and-their-applications>.
- [39] Dic. de 2023. dirección: <https://www.freertos.org/index.html>.
- [40] *Pololu Robotics and Electronics*. dirección: <https://www.pololu.com/product/975>.
- [41] N. Gourier, D. Hall y J. L. Crowley, *Estimating Face orientation from Robust Detection of Salient Facial Structures*. dirección: <http://crowley-coutaz.fr/jlc/papers/Pointing04-Gourier.pdf>.
- [42] Dirección: <https://www.deviantart.com/no-stars/art/Artists-anatomy-reference-male-head-angles-740723836>.

CAPÍTULO 15

Anexos

15.1. Códigos

Para poder acceder a la lista de códigos necesarios para la implementación de la interfaz gráfica que permite el manejo de simulaciones y pruebas físicas para controlar agentes robóticos móviles por medio de posiciones y gestos de la cabeza, ingresar al siguiente link: https://github.com/fue19389/CODIGOS_TESIS/tree/graphMODELS2_ANALOG

15.2. Videos demostrativos

Para observar a la demostración de la captura de datos acceder al siguiente link: <https://youtu.be/zi-wfWPRmNw>

Para acceder a la primera demostración del funcionamiento del reconocimiento de gestos y posiciones de la cabeza por medio de implementaciones físicas acceder al siguiente link: <https://youtu.be/BH5b8ZK79zQ>

Para acceder a la segunda demostración del funcionamiento del reconocimiento de gestos y posiciones de la cabeza por medio de implementaciones físicas acceder al siguiente link: <https://youtu.be/UJ2htHZI7T8>

