PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

DISEÑO DE UN SISTEMA DE LOCALIZACIÓN DE UN ROBOT MÓVIL BASADO EN MAPEO SIMULTÁNEO

Tesis para obtener el título profesional de Ingeniero Electrónico

AUTOR:

Jhomer Rodrigo Contreras Paucca

ASESOR:

Dr. Carlos Gustavo Pérez Zuñiga

Lima, mayo de 2022

A mis padres y a mis hermanos por
ser los mayores soportes en mi vida universitaria.

El autor quiere agradecer al CONCYTEC y su programa PROCIENCIA por brindar los fondos para el desarrollo de esta tesis, que forma parte del proyecto 160-2020 - Robot Móvil Teleoperado para Manejo de Afecciones de Salud Mental en Pacientes con Enfermedades Infecciosas.

RESUMEN

El trabajo de investigación presente tiene como objetivo el diseño de un sistema de localización de un robot móvil humanoide dentro de los ambientes de un centro de salud, específicamente que le permita al robot desplazarse en una habitación y posicionarse a una distancia adecuada frente al paciente que se encuentra en estado de reposo en una camilla de hospital. Para alcanzar este objetivo, este sistema se basa en la clasificación de los elementos percibidos en una habitación a través de una cámara y un computador de alto rendimiento, mediante el cual se resuelve un problema de clasificación en la imagen adquirida al detectar al paciente y la camilla como objetivos, luego se realiza un proceso de acercamiento del robot a estos objetivos hasta poder reconocer el rostro de una persona.

En el primer capítulo, se mencionan las bases teóricas que comprende la localización de un robot móvil basada en mapeo simultáneo, métodos SLAM y la relación de ambos con el objetivo del proyecto. En el segundo capítulo, se describe la metodología de los algoritmos basados en redes neuronales convolucionales y su base teórica, con el fin de determinar un algoritmo robusto y novedoso a beneficio del proyecto; adicionalmente se incluye una propuesta de solución explicando las etapas de dicho algoritmo. Posteriormente, en el tercer capítulo se desarrolla el diseño del algoritmo y su estructura mediante librerías y codificación realizadas de acuerdo al lenguaje de programación compatible con un computador de alto rendimiento. En el último y cuarto capítulo, se realizan resultados experimentales y se muestran los resultados de ellos. Finalmente, se presentan las conclusiones obtenidas en todo el proceso y recomendaciones. La presente tesis forma parte del proyecto CONCYTEC 160-2020 - Robot Móvil Teleoperado para Manejo de Afecciones de Salud Mental en Pacientes con Enfermedades Infecciosas.

ÍNDICE

RESUMEN iv

ÍNDICE v

LISTA DE FIGURAS vii

LISTA DE TABLAS ix

INTRODUCCIÓN x

CAPITULO 1: MARCO PROBLEMÁTICO DEL ANÁLISIS DE LA LOCALIZACIÓN DE UN ROBOT BASADO EN MAPEO SIMULTÁNEO 1

1.1 PROBLEMÁTICA DE LA INVESTIGACIÓN 1
1.2 PRESENTACIÓN DEL ASUNTO DE ESTUDIO 1
1.3 ESTADO DEL ARTE 3
1.3.1 Métodos SLAM 4
1.3.2 Reconocimiento facial 8
1.3.3 Cámaras 10
1.4 ROBOT MÓVIL TELEOPERADO PARA MANEJO DE AFECCIONES MENTALES EN PACIENTES CON ENFERMEDADES INFECCIOSAS 12
1.5 JUSTIFICACIÓN 13
1.6 OBJETIVOS 13
1.6.1 Objetivo General 13
1.6.2 Objetivos Específicos 13

CAPITULO 2: MARCO TEÓRICO 14

2.1 GENERALIDADES 14
2.2 CLASIFICACIÓN DE IMÁGENES 14
2.3 REDES NEURONALES CONVOLUCIONALES 15
2.3.1 Neurona 15
2.3.2 Red neuronal artificial 15
2.3.3 Arquitectura de las CNN 16
2.4 R-CNN 17
2.4.1 *Fast* R-CNN 18
2.4.2 *Faster* R-CNN 18
2.5 *YOU ONLY LOOK ONCE (YOLO)* 18

2.6 PARÁMETROS DE MOVIMIENTO 19
2.6.1 Geometría de la cámara 19
2.6.2 Estimación de la posición 20
2.7 MODELO SOLUCIÓN 21

CAPITULO 3: DISEÑO DE LOS ALGORITMOS 23

3.1 ADQUISICIÓN DE LA IMAGEN 23
3.2 BALANCE DE POTENCIA DE CONSUMO DEL SISTEMA 25
3.3 IMPLEMENTACIÓN DEL ALGORTIMO YOLOv4 26
3.4 EXTRACCIÓN DE LOS PARÁMETROS Y CREACIÓN DEL DATASET 28
3.4.1 Estimación de dirección 31
3.4.2 Estimación de distancia 31
3.5 DESARROLLO DEL SISTEMA 31
3.6 METODOLOGÍA PARA LA VALIDACIÓN 35
3.7 DESCRIPCIÓN DE LA APLICACIÓN DEL SISTEMA DE LOCALIZACIÓN 35

CAPITULO 4: PRUEBAS EXPERIMENTALES Y RESULTADOS 38

4.1 POSICIONAMIENTO DEL ROBOT EN LA PUERTA DE LA HABITACIÓN 38
4.2 SISTEMA DE LOCALIZACIÓN 39
4.3 IMPLEMENTACIÓN DEL ALGORITMO EN EL SISTEMA 40
4.4 VALIDACIÓN DEL SISTEMA DE LOCALIZACIÓN EN EL HOSPITAL REZOLA DE CAÑETE 41
4.5 ANÁLISIS DE LOS RESULTADOS 46

CONCLUSIONES DE LOS RESULTADOS 49

RECOMENDACIONES 50

REFERENCIA BIBLIOGRÁFICA 51

ANEXOS 54

LISTA DE FIGURAS

Figura 1.1: Gráfica de los principales parámetros SLAM [5] 3

Figura 1.2: Robot humanoide ARI [9] 4

Figura 1.3: Robot humanoide XR-1 [14] 6

Figura 1.4: Robot humanoide Tokyo [17] 8

Figura 1.5: e-CAM80_CUNX [19] 10

Figura 1.6: e-CAM131_CUNX [20] 11

Figura 1.7: e-CAM82_USB [21] 12

Figura 1.8: Visualización del robot para afecciones mentales 12

Figura 2.1: Modelo de clasificación de imágenes [22] 14

Figura 2.2: Modelo artificial de una neurona [24] 15

Figura 2.3: Modelo artificial de una red neuronal [25] 16

Figura 2.4: Arquitectura de las redes neuronales convolucionales [27] 17

Figura 2.5: Geometría de la cámara [22] 19

Figura 2.6: Cuadro delimitador dentro de la imagen [32] 20

Figura 2.7: Diagrama de bloques del modelo solución 21

Figura 2.8: Diagrama de flujo del modelo solución 22

Figura 3.1: Componentes del sistema de localización 23

Figura 3.2: Diagrama de conexiones de componentes electrónicos del sistema de localización 24

Figura 3.3: Modelamiento de la cámara y su rango de visión 25

Figura 3.4: Conexiones para la alimentación del sistema 25

Figura 3.5: YOLOv4 y sus características principales 27

Figura 3.6: Salida "*.json*" del algoritmo YOLOv4 27

Figura 3.7: Módulo fabricado para la toma de parámetros y creación del dataset 29

Figura 3.8: Diagrama de generación del modelo predictivo 29

Figura 3.9: Diagrama de bloques del estimador de dirección 32

Figura 3.10: Diagrama de la red neuronal del estimador de dirección 32

Figura 3.11: Diagrama de bloques del estimador de distancia. 33

Figura 3.12 Diagrama de la red neuronal del estimador de distancia 34

Figura 3.13: Ejemplo de estimación en la salida generada por el algoritmo 36

Figura 4.1: Plano de piso donde se realizaron las pruebas Hospital Regional de Cañete 38

Figura 4.2: Mapa generado por el robot y ubicación inicial para las pruebas 39

Figura 4.3: Diagrama de conexiones del sistema de localización 40

Figura 4.4: Diagrama de bloques del sistema de localización 40

Figura 4.5: Primera secuencia de imágenes adquiridas por el sistema 42

Figura 4.6: Máximos y mínimos de detcción por categoría 43

Figura 4.7: Segunda secuencia de imágenes adquiridas por el sistema 44

Figura 4.8: Resumen de la operación real del sistema 45

Figura 4.9: Precisión del entrenamiento y validación del modelo de orientación 46

Figura 4.10: Pérdidas del entrenamiento y validación del modelo de orientación 47

Figura 4.11: Precisión del entrenamiento y validación del modelo de distancia 47

LISTA DE TABLAS

Tabla 1.1: Comparación entre los métodos SLAM. Elaboración propia. 7

Tabla 1.2: Comparación entre las principales características de robots que presentan navegación autónoma basada en SLAM. Elaboración propia. 9

Tabla 1.3: Comparación entre las cámaras seleccioandas. Elaboración propia. 12

Tabla 3.1: Descripción del dataset del estimador de dirección. Elaboración propia. 30

Tabla 3.2: Descripción del dataset del estimador de distancia. Elaboración propia. 30

Tabla 3.3: Descripción de la aplicación del sistema. Parte1. Elaboración propia. 36

Tabla 3.4: Descripción de la aplicación del sistema. Parte2. Elaboración propia. 37

Tabla 3.5: Descripción de la aplicación del sistema. Parte3. Elaboración propia. 37

INTRODUCCIÓN

En los tiempos actuales, el cuidado de la salud mental requiere de una especial atención, por ejemplo, un caso particular, es el de los pacientes que padecen de enfermedades infecciosas y requieren atención psicológica. La protección del personal de salud mental para evitar contagios es vital para que los especialistas puedan realizar adecuadamente su trabajo. Es así que, la comunidad científica mundial viene introduciendo nuevas tecnologías por ejemplo en la atención de pacientes que salen de cuidados intensivos, con procesos lentos de recuperación, y que presentan cuadros de estrés, depresión, ansiedad, entre otros. En este contexto, investigadores de la PUCP vienen desarrollando el proyecto: Robot Móvil Teleoperado para Manejo de Afecciones de Salud Mental en Pacientes con Enfermedades Infecciosas, que permitirá atención de pacientes por telepresencia.

Específicamente esta tesis se enfoca en el diseño de un algoritmo de localización e identificación de pacientes que permitirá que el robot de teleoperación pueda posicionarse adecuadamente frente al paciente para iniciar procesos de terapias y rehabilitación; para ello, se describen los algoritmos de localización y mapeo simultáneo de los robos móviles teleoperados desarrollados en la última década, además de métodos de clasificación de imágenes y estimación de trayectoria para la localización de pacientes.

CAPÍTULO 1: MARCO PROBLEMÁTICO DEL ANÁLISIS DE LA LOCALIZACIÓN DE UN ROBOT BASADO EN MAPEO SIMULTÁNEO

1.1 PROBLEMÁTICA DE LA INVESTIGACIÓN

El 11 de marzo del 2020, la Organización Mundial de la Salud (OMS) declaró oficialmente la nueva enfermedad del SARS-CoV-2 (COVID-19) una pandemia. Después de un mes, el 14 de abril de 2020, se registraron dos millones de casos, de los cuales el 6% representaba la cantidad de fallecidos. Con el fin de limitar la propagación del COVID-19 se establecieron medidas estrictas de salud pública a nivel mundial como la cuarentena, el distanciamiento y aislamiento social. En esta emergencia sanitaria, los centros de salud se enfocaron en el tratamiento de miles de pacientes que padecen de esta enfermedad infecciosa, pero carecen de un sistema de respuesta hacia los cuadros de depresión, ansiedad, insomnio, entre otras afecciones mentales que también sufren estos pacientes durante el tratamiento de dicha enfermedad [1]. Esta problemática evidencia un sistema ineficiente para el tratamiento de personas con afecciones mentales que presentan enfermedades infecciosas, es por ello que se necesita de nuevas tecnologías para la creación de un protocolo de atención en tales casos.

Además, como consecuencia del confinamiento y la coyuntura social durante la pandemia, se evidenció un aumento a un 56% en ansiedad generalizada y un 30% en trastornos depresivos a lo largo del periodo de pandemia [2]. Asimismo, para los pacientes que se encuentran internados en los hospitales y sufren de COVID-19 el aumento de estrés o algún desorden mental es aún mayor [1].

Es por ello que existe la necesidad de plantear soluciones para atender pacientes que salen de cuidados intensivos, tienen procesos lentos de recuperación, y presentan cuadros de estrés, depresión, ansiedad, entre otros. Dado este entorno, el desarrollo de robots teleoperados que puedan interactuar con estos pacientes presenta un gran avance en el campo del tratamiento psicológico, sin embargo, dada la novedad científica presenta diferentes retos a resolver.

1.2 PRESENTACIÓN DEL ASUNTO DE ESTUDIO

Actualmente, la robótica teleoperada es un tema de mucho interés por parte de los investigadores en salud, ya que proporciona la opción de realizar el mismo trabajo humano, pero de manera remota. Como es sabido, el ser humano siempre ha buscado la manera de sintetizar el trabajo, establecer tecnologías que le eviten exponerse a situaciones peligrosas. Durante estos últimos años, se vienen desarrollando diversos robots con el fin de cumplir tareas específicas en distintas áreas de trabajo; por ejemplo,

en las áreas de logística, medicina, transporte público y atención de personal. De manera particular, en el área de la medicina, se están desarrollando robots para el monitoreo de pacientes, asistencia personalizada y operaciones quirúrgicas.

Específicamente para el tratamiento de pacientes que presentan afecciones mentales, se requiere diseñar un sistema que permita posicionar a un robot frente a un paciente en la habitación que se encuentre considerando los diversos objetos presentes que alteren la trayectoria; uno de los métodos más empleados para resolver esta tarea es el uso de algoritmos de localización y modelación simultánea [2]. Por otro lado, es recomendable el uso de redes neuronales convolucionales para clasificadores de objetos, mediante estas redes se puede implementar un sistema de reconocimiento de personas dentro de una habitación y determinar si esta es un paciente o no.

RECONOCIMIENTO DE PERSONAS

Entre las tecnologías para la identificación de rostros y verificación biométrica, el reconocimiento facial es ampliamente utilizado, ya que es un método no invasivo. La robustez de un sistema de reconocimiento facial se limita al grado de postura de la cabeza que está involucrado; sin embargo, las tasas de reconocimiento de los sistemas de última generación caen significativamente para ángulos de postura grandes [3]. Especialmente para tareas en las que se espera una gran variación en la postura de la cabeza, como en la interacción humano-robot o la localización de un rostro en movimiento, el reconocimiento facial invariable es inevitable. Por otro lado, el aumento de los robots teleoperados de asistencia doméstica aumentará la demanda de algoritmos que puedan adaptarse a condiciones incontroladas [4]; por otro lado, si se quiere lograr un reconocimiento más certero, se hace el reconocimiento hasta la identificación de un rostro humano, es decir, que existe la posibilidad de lograr el reconocimiento de una persona mediante el cuerpo completo y de esta manera evitar complicaciones en la detección de rostros.

MAPEO Y LOCALIZACIÓN SIMULTÁNEA

Con la reciente tecnología y coyuntura actual, se ha dado un enfoque más autónomo a los futuros robots que se emplearán en el área de la medicina, por lo que una de las funciones básicas que debe tener este robot es de poder generar un mapa del lugar donde se encuentra para posteriormente recorrer en él. Como solución a este pedido, se emplea *Simultaneous Localization and Mapping* (SLAM) que es el proceso mediante el cual un robot puede construir un mapa de un entorno y al mismo tiempo utilizar este mapa para determinar su ubicación [5]. Es otros términos, que tanto la trayectoria como puntos de referencia se estiman a pesar de no tener la información a priori; de esta

manera al generar su propio mapa del entorno que lo rodea se podrá enviar información de la localización del paciente para dirigirse hacia él.

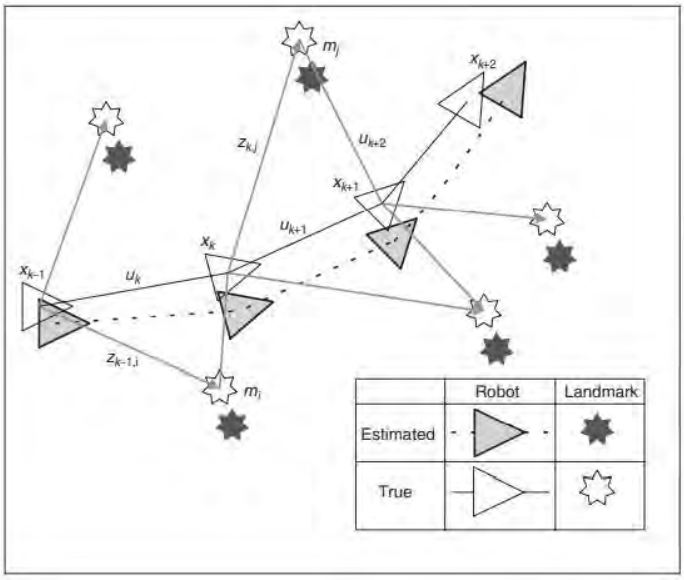


Figura 1.1 Gráfica de los principales parámetros SLAM. [5].

En la Figura 1.1 se observa el principal objetivo de los métodos SLAM, el cual es lograr una estimación aproximada a la posición real del robot móvil en un cualquier instante de tiempo de acuerdo a los puntos de referencias seleccionados por el algoritmo. La figura presentada describe a los puntos de referencia $m_{i,j}$ como las posiciones reales de los puntos que se toman como referencia en el entorno del robot móvil y el mismo símbolo en oscuro, define las aproximaciones de estos según el algoritmo; asimismo se observa los términos x_{k-1}, x_k, x_{k+1} que hace referencia a la posición real del robot móvil y el mismo símbolo en oscuro, define las posiciones estimadas en el instante de tiempo k donde se quiere evaluar la posición del robot móvil aplicando los algoritmos SLAM [5].

1.3 ESTADO DEL ARTE

En esta sección se hará una revisión del estado actual del uso algoritmos de SLAM y reconocimiento facial en robots de interacción humana, destacando la complejidad y el tiempo de procesamiento de los métodos que puedan contribuir al diseño de un algoritmo eficiente. En primer lugar, se recopilará información sobre los métodos SLAM existentes y su uso en las recientes tecnologías de robots autónomos. En segundo lugar, se mostrarán los métodos de reconocimiento de objetos y personas encontrados en artículos o investigaciones recientes y se realizará una comparación por cada uno. Finalmente, se presentarán algunas alternativas de cámaras comerciales donde se puedan integrar ambos tipos de métodos.

1.3.1 Métodos SLAM

SLAM es un proceso mediante el cual un robot puede construir un mapa de su entorno y al mismo tiempo utilizar este mapa para determinar su ubicación. A continuación, se presenta los principales métodos SLAM para elegir uno adecuado para el sistema de localización de la persona y navegación autónoma.

EKF-SLAM : Método no lineal que describe el movimiento del móvil y el modelo observacional mediante la geometría y cinemática. En robótica, EKF SLAM es una clase de algoritmos que utiliza el filtro Kalman extendido (EKF) para localización y mapeo simultáneos. Normalmente, los algoritmos que aplican este método se basan en las características del algoritmo de máxima verosimilitud para la asociación de datos [5], [6].

En la práctica, es poco usado en robots de navegación autónoma debido a que el paso de actualización del sensor se puede hacer computacionalmente manejable con cualquier variedad aproximada de métodos EKF [7].

ORB-SLAM: Es una solución SLAM versátil y precisa para cámaras monoculares, estéreo y RGB-D. Es capaz de calcular en tiempo real la trayectoria de la cámara y una escasa reconstrucción en 3D. Es un sistema de localización y mapeo simultáneo monocular (SLAM) basado en características que opera en tiempo real, en ambientes interiores y exteriores pequeños y grandes. El sistema es resistente al desorden de movimiento severo, permite el cierre y la relocalización del bucle de línea de base amplia e incluye una inicialización automática completa [8].

Figura 1.2. Robot humanoide ARI [9].

En la Figura 1.2 se observa el robot ARI de la empresa PAL Robotics. El cual es capaz de realizar funciones para asistencia en hospitales, atención al público, guía y entretenimiento en salas de espera. Respecto como logra la navegación autónoma, el robot ARI usa el método ORB-SLAM como una parte principal de su algoritmo [9].

FAST-SLAM: Método no lineal que aproxima razonablemente mediciones de distancia conociendo la posición del vehículo, elabora un mapa de trayectoria de acuerdo al modelo probabilístico de la ubicación. Se basa en la combinación de un filtro de partículas sobre un mapa determinado por un conjunto de filtros de Kalman y utiliza distribuciones gaussianas de dos dimensiones [10]. Su uso en robots sigue la lógica planteada:

- Probar una nueva ruta de robot dado el nuevo control
- Actualizar los filtros de puntos de referencia correspondientes a la nueva observación.
- Asignar un peso a cada una de las partículas
- Muestrear nuevamente las partículas según su peso.

HECTOR-SLAM: Método basado en el escaneo láser de alta resolución y alta frecuencia e ignora la información que puede proporcionar la odometría de las ruedas del móvil. Además, el método utilizado para realizar el mapeo para un entorno desconocido en este algoritmo es el mapeo de cuadrícula y se localiza mediante la búsqueda de coincidencias [11]. En la posición inicial, los datos del primer cuadro que proporciona LiDAR se utilizan directamente para construir un gráfico, luego los datos del sensor se comparan con el mapa y se deriva una pose óptima de la unidad LiDAR. Adicionalmente, este método es presentado en variedades de robots móviles implementados en ROS [11], [12].

VSLAM: Los sistemas SLAM visuales existentes se dividen generalmente en dos categorías: directos y basados en funciones. Los sistemas SLAM directos minimizan los errores fotométricos para resolver las poses de la cámara y construir mapas 3D. Los SLAM basados en características detectan puntos clave o segmentos de línea en imágenes y los combinan para la localización y el mapeo. Los métodos directos pueden obtener mapas 3D relativamente densos, pero sufren variaciones de intensidad en las secuencias de imágenes. Además, no pueden realizar el cierre de bucle debido al extenso cálculo entre cuadros y los altos requisitos de memoria [13]. Por el contrario, los métodos basados en características son más robustos a la variación de intensidad y apoyan el cierre del bucle; generalmente en la práctica este algoritmo se usa conjuntamente con el ORB-SLAM [12], [13].

En la Figura 1.3 se muestra al robot humanoide XR-1 de la empresa Cloud Brain. El cual es capaz de realizar manipulaciones robóticas controladas por visión, como sostener objetos, abrir puertas y enhebrar una aguja, lo que lo hace perfecto en aplicaciones como conserje, recepcionista, guía de negocios o personal de servicio VIP

en diferentes escenarios. Para la navegación automática emplea el método VSLAM y para ello usa LiDAR, cámaras y múltiples sensores adicionales [14].

Figura 1.3. Robot humanoide XR-1 [14].

Tabla de comparación de métodos SLAM

Se presenta a continuación la tabla de comparación como resumen del estudio intensivo de los métodos más convenientes para generar algoritmos SLAM con el fin de diseñar un sistema de navegación autónomo y robusto ante cambios dinámicos en el entorno para un robot teleoperado capaz de localizar y dirigirse hacia una persona. Según la Tabla 1.1, se muestra las características principales de los métodos SLAM y se determina que el EKF-SLAM y FAST-SLAM tienden a ser más probabilísticos que los otros métodos que usan más tipos de sensores, ya que estos eran algoritmos iniciales SLAM que se implementaron en un robot móvil para evasión de obstáculos [10]. Además, para objetivos del proyecto se requiere un método novedoso que cuente con una estimación tanto de posición como puntos de referencias certera. Por otro lado, el HECTOR-SLAM presenta mayor exactitud que los dos anteriores mencionados, sin embargo, se usa normalmente para monitoreo y creación de mapa de entorno, ya que en su estimación no considera la odometría de las ruedas que es un factor determinante para los robots teleoperados terrestres [10].

Es por ello que la mayoría de robots de navegación autónoma para espacios cerrados usan algoritmos VSLAM como se muestra en la Tabla 1.2 o combinaciones con esta. Por lo tanto, considerando la función dentro de un hospital, el método SLAM que se buscará implementar será un método semejante a la combinación de tipo ORB y VSLAM, tomando como referencia el robot ARI de la empresa PAL Robotics. Uniendo VSLAM (toma de puntos de referencia) y la odometría en el entorno de trabajo de sistema operativo robótico (ROS), el cual es un conjunto de bibliotecas de software y

herramientas que ayudan a crear aplicaciones de robótica, existe el método GMapping, que combina métodos SLAM y además puede ser simulado en diferentes softwares para su validación, por lo que es el más conveniente para esta aplicación.

Tabla 1.1 Comparación entre los métodos SLAM. [6], [8], [10]–[13], [15].

	Ventajas	**Desventajas**	**Sensor**
EKF-SLAM	• El vector de estado está compuesto por la ubicación de la entidad y algunos elementos del mapa, estimados de forma recursiva a partir de los modelos no lineales de observación y transición.	• Respecto al vector de estado, el algoritmo sufre una complejidad que depende directamente solo del número de puntos de referencia.	Sensor de alcance 2D
ORB-SLAM	• Relocalización robusta de la velocidad de fotogramas y detección de bucles. • Invariantes a la rotación y la escala lo que da como resultado un reconocedor rápido con invariancia al punto de vista.	• Se requiere un procedimiento para crear un mapa inicial porque la profundidad no se puede recuperar de una sola imagen o presenta una alta incertidumbre.	Cámara monocular Cámara estereoscópica
FAST-SLAM	• Respecto al vector de estado, el algoritmo sufre una complejidad que depende directamente de una constante, definida por el número de partículas, y el número de puntos de referencia.	• Se estima la asociación para cada partícula lo cual genera un entorno grande de búsqueda en el espacio de las posibles trayectorias y asociaciones de datos del móvil.	Sensor de alcance 2D
HECTOR-SLAM	• No es necesario utilizar el método de detección de cierre de bucle ya que el método puede completar con precisión el bucle en muchas escenas reales.	• Registra la posición óptima del LiDAR como enlace base ya que no aplica odometría en su algoritmo, lo que resulta disminuye su exactitud.	LiDAR 2D
VSLAM	• Utiliza las esquinas como puntos de referencia debido a sus características invariables y su amplio estudio en el contexto de la visión por computadora.	• La asociación de datos presenta problemas en casos particulares como la detección de cierre de bucle, ubicación inicial arbitraria, mapeo multisesión y cooperativo.	Sensor ultrasónico LiDAR Cámara monocular

1.3.2 Reconocimiento Facial

Los métodos de reconocimiento facial, realizan su ejecución mediante una cámara y algoritmos realizados por un procesador. Estos algoritmos se basan en el reconocimiento del rostro, extracción de la figura del rostro y el relacionamiento de acuerdo a las coincidencias con la persona que se quiera reconocer [4].

Reconocimiento del rostro: Se usa métodos como aumento de datos, duplicar, rotar, aplicar filtros a una imagen o crear un modelo 3D de la cara para mejorar el rendimiento del sistema de reconocimiento antes situaciones que pueden resultar perjudiciales como poses específicas, iluminaciones, expresiones y oclusiones [16].

Extracción de la figura: La extracción de características es la tarea de obtener características de identidad robustas y discriminatorias de un rostro. Es el componente central de la mayoría de los algoritmos de reconocimiento facial, ya que es necesario tanto para la verificación facial como para la identificación facial [16].

Relacionamiento de rostro: La coincidencia facial es el último paso del proceso de reconocimiento facial y tiene la tarea de hacer coincidir la nueva muestra con una identidad conocida dentro de la base de datos de rostros [16].

En la Figura 1.4 se observa al robot humanoide de la empresa ADD. Tiene la capacidad de recibir personas por su nombre, saludar a residentes en función de la hora y el día mediante con un sistema de reconocimiento facial. Capaz de ser teleoperado y realizar videollamadas y conferencias. Además, tiene la capacidad de navegación autónoma mediante algoritmos SLAM.

Figura 1.4. Robot humanoide Tokyo [17].

Tabla de comparación general de características principales

En la Tabla 1.2 se muestra una comparación de acuerdo a los robots humanoides presentados anteriormente, con el fin de evaluar los requerimientos a tomar en cuenta para el diseño y validación del algoritmo, tales como altura, tipo de método, tipo de cámara usada y la comunicación que utiliza con el desarrollador propio de cada uno de los robots para su respectiva implementación en el proyecto.

Tabla 1.2. Comparación entre las principales características de robots que presentan navegación autónoma basada en SLAM. [9], [14], [17], [18].

	ARI	**XR-1**	**Tokyo**
Método SLAM	VSLAM ORB-SLAM	VSLAM	VSLAM
Telepresencia	SI	NO	SI
Navegación autónoma	SI	SI	SI
Carga automática	SI	NO	NO
Reconocimiento de personas	NO	NO	SI
Horas de duración	8-12	8	6-8
Plataforma	NVIDIA Jetson TX2.	NVIDIA Jetson TX2.	Intel NUC i5-7260U NVIDIA
Altura (cm)	165	158	165
Cámara	8.0MP RGB	8MP (4K)	8MP (4K)
Pantalla	10.1 pulgadas 1200x800	NO	10.1 pulgadas HD
Batería	24V/40Ah	36V/40Ah	-

Se observa en la Tabla 1.2 que las mejores características respecto a navegación autónoma corresponden al robot ARI, con lo métodos VSLAM y ORB-SLAM; por otro lado, respecto al reconocimiento de personas el robot Tokyo es el único que tiene esta función. Además, de acuerdo a la tabla se emplearon una cámara de 8MP para la captura de imágenes en tiempo real, el cual puede usarse en el reconocimiento de personas, es por ello que en la siguiente sección se evalúan las opciones de cámaras.

1.3.3 Cámaras

Los métodos de reconocimiento facial necesitan de una cámara para la evaluación el desarrollo de su algoritmo, la cual manda información acerca de los objetos de su entorno.

Primeramente, se establece los parámetros para la elección de la cámara con la que se trabajará el reconocimiento facial y el algoritmo de localización. De los cuales los principales son, resolución del dispositivo, consumo de energía, cuadros por segundo (*frames per seconds*) y el tipo de sensor que se utiliza [3], [4].

Una tarea que implique reconocimiento facial, como la búsqueda de un individuo específico, podría ser engañada por estos reconocimientos positivos no deseados, provocando ramificaciones impredecibles en el comportamiento de este robot .

Debido a que para el robot de afecciones mentales se usa un procesador Jetson-Xavier-NX se debe de utilizar una cámara la cual es compatible con este dispositivo, para ello entre las mejores opciones se encontraron las siguientes:

e-CAM80_CUNX

e-CAM80_CUNX es una cámara MIPI CSI-2 *Ultra-lowlight* de 8MP capaz de transmitir 4K a 44 fps. Esta cámara de 8MP está basada en el sensor de imagen CMOS SONY STARVIS ™ IMX415. Esta cámara 4K de Sony se puede conectar directamente con el kit de desarrollador NVIDIA® Jetson Xavier NX y utiliza el procesador de señal de imagen (ISP) Jetson Xavier NX integrado de NVIDIA® para realizar todas las funciones automáticas (balance de blancos automático, control de exposición automática). (e-con Systems, 2021)

Figura 1.5. e-CAM80_CUNX. [19].

Según el datasheet del dispositivo, este puede llegar hasta los 3840(H) x 2160(V) de resolución a 44 fps y una potencia de consumo es de 0.7 W.

e-CAM131_CUNX

e-CAM131_CUNX es una cámara en color MIPI de enfoque fijo 4K Ultra HD para el kit de desarrollador NVIDIA Jetson Xavier NX / NVIDIA Jetson Nano. e-CAM131_CUNX se basa en el sensor de imagen CMOS AR1335 de 13MP de ON Semiconductor. Tiene un chip procesador de señal de imagen (ISP) dedicado, que realiza todas las funciones automáticas como balance de blancos automático y control de exposición automática. (e-con Systems, 2021)

Comparándola con el anterior modelo, esta presenta mejor la resolución máxima que se puede obtener, pero una gran desventaja frente a los cuadros por segundos (fps). Por otro lado, según el datasheet del dispositivo, este puede llegar hasta los 4192(H) x 3120(V) de resolución a 10 fps y una potencia de consumo es de 1.65 W [20].

Figura 1.6. e-CAM131_CUNX. [20].

e-CAM82_USB

e-CAM82_USB es una cámara 4K USB UVC *Ultra-lowlight* basada en el sensor de imagen SONY STARVIS® IMX415 CMOS de 1 / 2.8 ". Esta cámara 4K ultra-HD USB 2.0 tiene un procesador de señal de imagen (ISP) integrado y dedicado que realiza todas las funciones automáticas (balance de blancos automático, exposición automática). (e-con Systems, 2021).

Comparando este dispositivo con los anteriores presentados, tiene una ventaja en el tipo de conexión (USB) que presenta para su comunicación con la Jetson Xavier NX. Si bien no llega hasta los 13MP que puede ofrecer la eCAM-131_CUNX, para los algoritmos planteados es suficiente con uno de 8MP, el cual si es posible obtener de este modelo. Entonces, el dispositivo puede llegar hasta los 3840(H) x 2160(V) de resolución a 35-15 fps y una potencia de consumo es de 1.07 W según lo indicado en su datasheet [21].

Figura 1.7 e-CAM82_USB. [21].

Tabla de comparación entre las cámaras

Tabla 1.3. Comparación entre las cámaras seleccioandas. [19]–[21].

Cámara	Resolución requerida	fps	Potencia de consumo	Conexión
e-CAM80_CUNX	4K UHD (3840 x 2160)	44	0.70 W	Cable Flex
e-CAM131_CUNX	4K (3840 x 2160)	15	1.65 W	Cable Flex
e-CAM82_USB	4K UHD (3840 x 2160)	30-15	1.07 W	USB

1.4 ROBOT MÓVIL TELEOPERADO PARA MANEJO DE AFECCIONES DE SALUD MENTAL EN PACIENTES CON ENFERMEDADES INFECCIOSAS

El proyecto cuenta con un conjunto de tecnologías en salud reflejado en el sistema de navegación, movimiento de extremidades, software amigable con el paciente, además de la teleoperación para la conexión remota a esta. El robot tiene forma humanoide, una estatura similar a la de un peruano promedio y un aspecto amigable para lograr la atracción de la atención de los pacientes lo cual se refleja en la Figura 1.8.

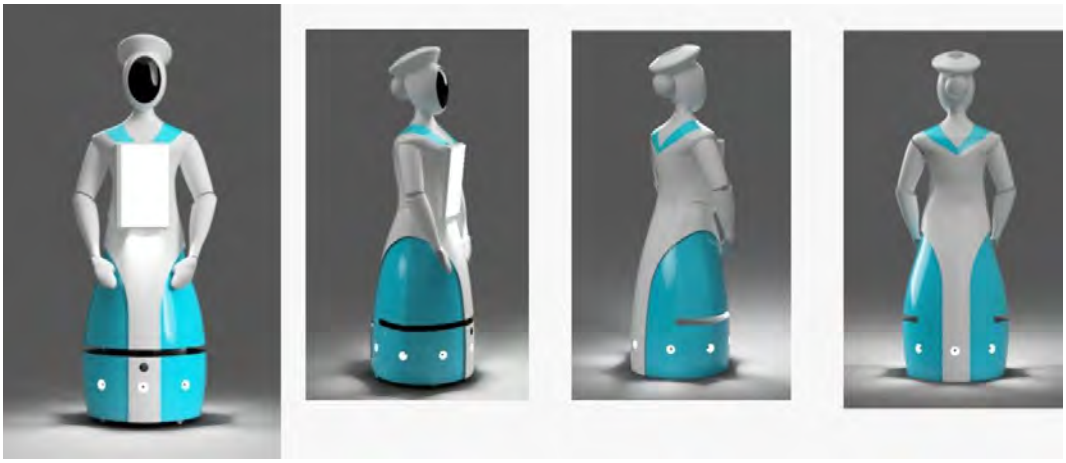


Figura 1.8 Visualización del robot para afecciones mentales.

1.5 JUSTIFICACIÓN

La salud mental es un campo de la medicina que requiere atención urgente dentro de la pandemia actual. Existe una gran necesidad de desarrollar soluciones para atender a los pacientes que salen de cuidados intensivos, con procesos lentos de recuperación, y que presentan cuadros de estrés, depresión, ansiedad, entre otros. Dado este entorno, mediante esta tesis se busca completar un sistema de localización de paciente del proyecto que se viene desarrollando conjuntamente el grupo GIT y GECA de la PUCP, Robot Móvil Teleoperado para Manejo de Afecciones de Salud Mental en Pacientes con Enfermedades Infecciosas, que permitirá atención de pacientes por telepresencia.

Específicamente esta tesis se enfoca en el diseño de un algoritmo de localización e identificación de pacientes que permitirá que el robot de teleoperación pueda posicionarse adecuadamente frente al paciente para iniciar procesos de terapias y rehabilitación. Lo cual permitirá potenciar los sistemas de salud para el tratamiento de pacientes aumentando la cantidad de pacientes atendidos y reduciendo drásticamente el riesgo de contagio.

1.6 OBJETIVOS

1.6.1 Objetivo General

- Diseñar un algoritmo que permita al robot de teleoperación reconocer al paciente y localizarse frente a él en un ambiente desconocido.

1.6.2 Objetivos Específicos

- Estudiar el estado del arte de los sistemas de mapeo, localización simultánea y de reconocimiento facial.
- Diseñar el diagrama electrónico de conexiones entre los componentes de captura de imágenes, computador de alto rendimiento, alimentación y comunicación con el computador principal.
- Diseñar un sistema que incluya algoritmos de reconocimiento de paciente y camilla, localización del paciente con respecto al robot y reconocimiento facial.
- Realizar pruebas experimentales y de validación en un centro de salud de los algoritmos desarrollados.

CAPÍTULO 2: MARCO TEÓRICO

2.1 GENERALIDADES

El diseño de un sistema de localización permitirá al robot teleoperado reconocer al paciente dentro de una habitación asignada para aproximarse de manera segura a una distancia adecuada con el fin de poder establecer una comunicación en forma de videoconferencia entre terapeutas y pacientes posibilitando la de atención de los pacientes con enfermedades infecciosas que sufren afecciones mentales.

Previo a detallar el marco teórico se debe considerar que de acuerdo a la tabla de comparación presentada en la Tabla 1.3, se optó por emplear la cámara 4K USB UVC *Ultra-lowlight* de modelo e-CAM82 para el sistema de localización y reconocimiento del paciente, la cual estará acoplada al robot teleoperado, esta cámara ha sido utilizada para la toma de datos en un centro de salud real para las pruebas de entrenamiento y validación del sistema desarrollado.

En este capítulo se explicarán las aplicaciones más importantes de *Deep Learning* en el análisis de imágenes. Asimismo, se presentará una breve descripción de las redes neuronales convolucionales, como también las diferentes redes neuronales para la detección de objetos en tiempo real y los parámetros de movimiento del robot a realizar para su posicionamiento. Finalmente, se presentará el modelo solución de la problemática.

2.2 CLASIFICACIÓN DE IMÁGENES

La clasificación de imágenes consiste en la generación de etiquetas en una imagen completa dada la probabilidad que esta sea de un tipo en particular [22]. En la Figura 2.1 se puede observar que la clasificación de imágenes consta de un conjunto de capas y cada una de ellas emplea una función específica como extracción de borde, contraste, filtros, para finalmente obtener la capa de reconocimiento .

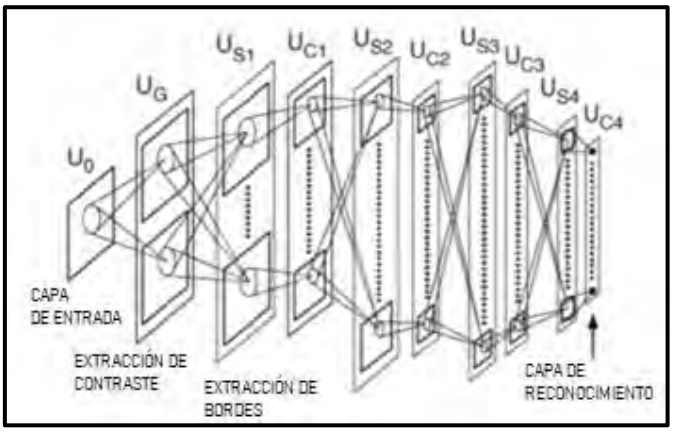


Figura 2.1: Modelo de clasificación de imágenes. [22]

2.3 REDES NEURONALES CONVOLUCIONALES (CNN)

Estas redes se emplean en imágenes bidimensionales, puesto que estas presentan fuertes dependencias espaciales en las regiones locales de la cuadrícula. Se usan para el reconocimiento o clasificación de imágenes [23].

Están formadas por neuronas que tienen pesos y sesgos que tienen la capacidad de aprender. Donde cada neurona recibe entradas, luego realiza un producto escalar y, opcionalmente, lo sigue con una no-linealidad. Toda la red expresa una única función de puntuación diferenciable: desde los píxeles de la imagen sin procesar en un extremo hasta las puntuaciones de la clase en el otro. Stanford University.

Para poder entender que son las CNN se definirán los conceptos de neurona y red neuronal.

2.3.1 Neurona

El modelo de una neurona artificial es una función cuyas entradas denominadas x_i, las cuales pueden ser los píxeles de una imagen, junto a los pesos w_i, forman una suma ponderada dentro del cuerpo de la neurona, a la cual se le añade un término de sesgo o bias denotado por b. Finalmente, la suma ponderada pasa por una función de activación que da como resultado la salida de la neurona, siendo el elemento principal de la CNN [24]. En la Figura 2.2 se observa el modelo explicado.

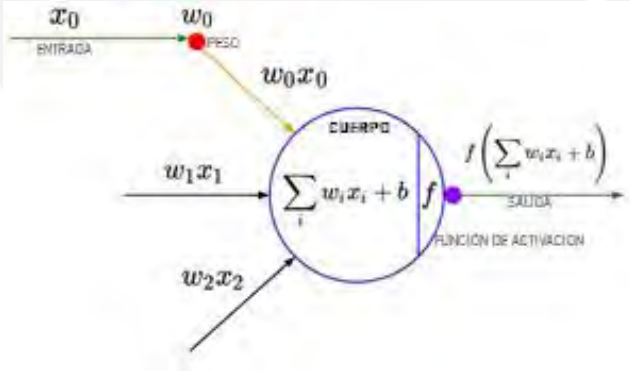


Figura 2.2: Modelo artificial de una neurona [24].

2.3.2 Red neuronal artificial

Las redes neuronales artificiales son modelos computacionales inspirados en el funcionamiento del cerebro que se entrenan según patrones de aprendizaje mediante prueba y error, al igual que lo hace el cerebro humano. Las redes neuronales suelen estar organizadas en capas, las cuales están formadas por una serie de nodos interconectados que contienen una función de activación [25]. Estos patrones se presentan a la red a través del *input layer* o capa de entrada, que se comunica con el

middle layer o capas intermedias donde el procesamiento real se realiza a través de un sistema de conexiones ponderadas. Las capas ocultas luego se vinculan al *output layer* o capa de salida donde se genera la respuesta como se muestra en la Figura 2.3.

Además, es una técnica para construir un programa de computadora que aprende de los datos. Se basa muy vagamente en cómo pensamos que funciona el cerebro humano. Primero, se crea una colección de neuronas de software y se conectan entre sí, lo que les permite enviarse mensajes entre sí. A continuación, se le pide a la red que resuelva un problema, lo que intenta hacer una y otra vez, cada vez fortaleciendo las conexiones que conducen al éxito y debilitando las que conducen al fracaso [26].

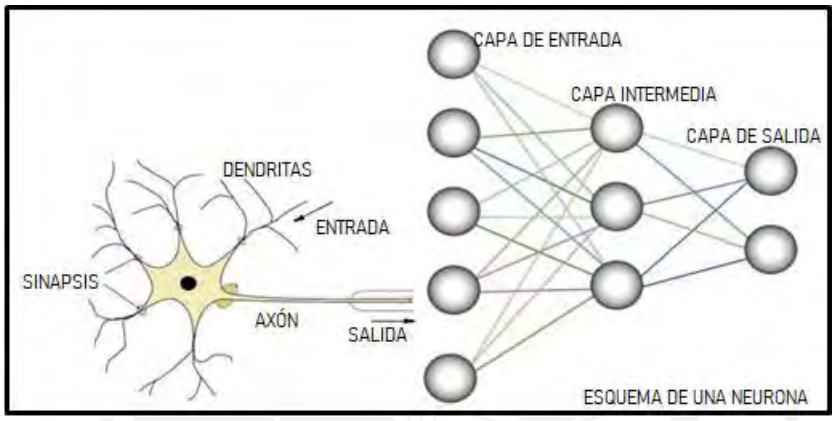


Figura 2.3: Modelo artificial de una red neuronal [25].

Se puede observar en la Figura 2.3 la semejanza entre la neurona biológica y el funcionamiento en la neurona artificial, en este caso con un tipo de arquitectura de redes *feed-forward*, donde la primera capa es la entrada y la última es la salida; en caso que exista una o más capas ocultas se les llaman redes neuronales profundas. Asimismo, para el análisis de las redes neuronales se considera el perceptrón como algoritmo de aprendizaje de las neuronas donde si los elementos del conjunto de entrenamiento son linealmente separables el algoritmo garantiza que se encontrará un conjunto de pesos que los clasifique correctamente a todos.

2.3.3 Arquitectura de las CNN

Las redes neuronales convolucionales aprovechan el hecho de que la entrada consiste en imágenes y restringen la arquitectura de una manera más sensible. En particular, a diferencia de una red neuronal normal, las capas de una red convoluciones tienen neuronas dispuestas en 3 dimensiones: ancho, alto, profundidad [24].

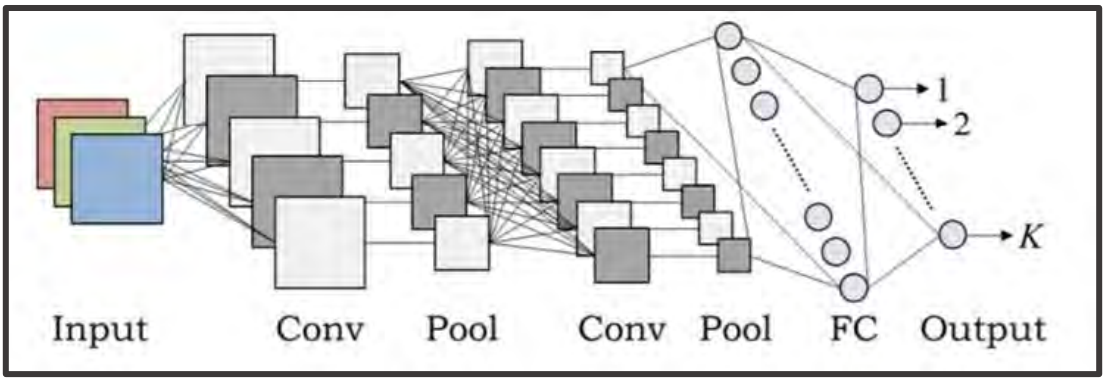


Figura 2.4: Arquitectura de las redes neuronales convolucionales [27].

En la Figura 2.4 se muestra gráficamente las principales capas de la red, que son la de convolucion, ReLu, *pooling* y *fully connected*.

- Convolución: En esta capa se extraen las características de la imagen de entrada por medio de la aplicación de filtros o *kernerls*. De esta operación se obtiene la salida denominada *feature map* a través de la convolución de una entrada con el *kernel* [27].
- ReLu: Es un tipo la función de activación que se usa comúnmente en las CNN, la cual se encarga de darle el carácter no lineal a la red. El aplicar esta función de activación modifica las dimensiones de las capas [27].
- *Pooling*: Esta capa tiene como función reducir el tamaño del *feature map* rectificado para que se tengan menos parámetros en la red mediante un submuestreo [27]. Por ejemplo, el tamaño de una imagen antes de aplicar *Pooling* es de 5x5 y después es 2x2. Es decir, baja la resolución de la imagen.
- *Fully Connected:* Después de convertir de matriz a vector se pueden formar capas totalmente conectadas como en el modelo convencional de una red. Si se aplica una función de activación, como sigmoide o Softmax se tendrá en la salida una cantidad de neuronas correspondiente a las clases [27].

2.4 R-CNN

El primer método a investigar en el robot de afecciones mentales es el R-CNN, el cual reduce la carga computacional en comparación a los algoritmos CNN que no usan un número definido de regiones propuestas, ya que realiza una búsqueda selectiva de solo 2000 regiones propuestas. Es decir, en lugar de intentar clasificar una gran cantidad de regiones, se puede trabajar regiones limitadas. Estas propuestas de regiones se generan utilizando el algoritmo de búsqueda selectiva, el cual consiste en generar una subsegmentación inicial, para generar regiones candidatas, posteriormente usar un

algoritmo dividido en tres módulos para combinar de forma recursiva regiones similares en regiones más grandes, finalmente usar las regiones generadas para producir las propuestas de regiones candidatas finales [28].

Entrando a detalle en esta búsqueda selectiva, el primer módulo genera propuestas regionales independientes de la categoría donde estas propuestas definen el conjunto de posibles detecciones disponibles para nuestro detector. El segundo es una gran red neuronal convolucional que extrae un vector de características de longitud fija de cada región. El tercer módulo es un conjunto de SVM lineales específicas de la clase [28].

2.4.1 Fast R-CNN

En lugar de enviar las propuestas de la región a la CNN, se envía la imagen de entrada a la CNN para generar un mapa de características convolucional. A partir del mapa de características convolucional, se identifica la región de las propuestas y se deforma en cuadrados y, mediante el uso de una capa de agrupación de la región de interés (RoI), se reforma en un tamaño fijo para que se pueda alimentar a una capa completamente conectada. A partir del vector de características RoI, se usa una capa *softmax* para predecir la clase de la región propuesta y también los valores de compensación para el cuadro delimitador [29].

2.4.2 Faster R-CNN

Consiste en incluir la proposición de regiones del R-CNN, el cual determina que regiones dentro de la imagen tienen mayor probabilidad de contener objetos, además de evitar introducir al clasificador regiones donde se puede notar que no existe presencia de personas, y por lo tanto se pueden incluir en un clasificador para un análisis para decidir si se trata de una persona la que se encuentra en la imagen [30].

2.5 YOU ONLY LOOK ONCE (YOLO)

Es un nuevo enfoque para la detección de objetos, el cual ha sido desarrollado recientemente. El trabajo previo en la detección de objetos reutiliza los clasificadores para realizar la detección. En cambio, este algoritmo enmarca la detección de objetos como un problema de regresión a cuadros delimitadores separados espacialmente y probabilidades de clase asociadas. Una sola red neuronal predice cuadros delimitadores y probabilidades de clase directamente a partir de imágenes completas en una evaluación. Dado que toda la línea de detección es una sola red, se puede optimizar de un extremo a otro directamente en la detección [31].

Esta distribución se refleja en la arquitectura unificada la cual es extremadamente rápida. YOLO procesa imágenes en tiempo real a 45 fotogramas por segundo. Una versión más pequeña de la red, Fast YOLO, procesa la asombrosa cantidad de 155 cuadros por segundo mientras logra el doble de precisión que otros detectores en tiempo real. En comparación con los sistemas de detección de última generación, YOLO comete más errores de localización, pero es menos probable que prediga falsos positivos en segundo plano [31].

2.6 PARÁMETROS DE MOVIMIENTO

En esta sección se definirán los métodos a emplear para poder estimar la posición de un objeto según las imágenes obtenidas por la cámara.

2.6.1 Geometría de la cámara

La cámara se define como una máquina recolectora de fotones e infiere valores de intensidad además de reductor de dimensiones, 3D a 2D. La información ambigua presente en una cámara sobre una imagen es la profundidad y no el color ni el sombreado (intensidad) [22]. Para nosotros cada ojo percibe una imagen 2D las cuales al combinarse generan la percepción de profundidad.

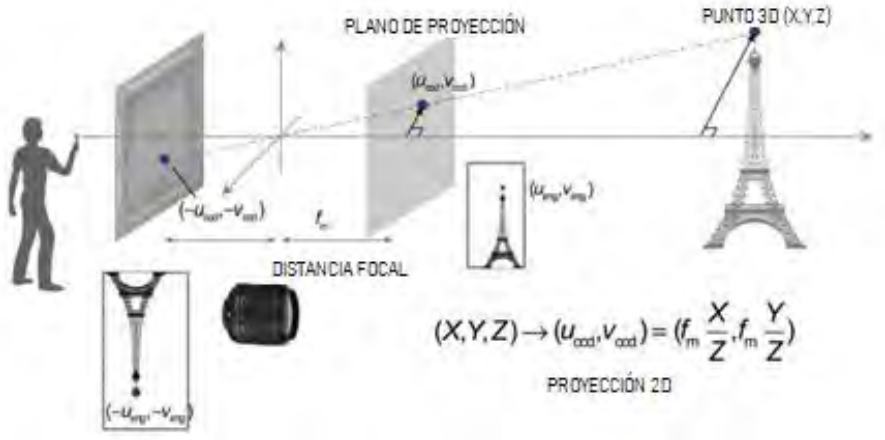


Figura 2.5: Geometría de la cámara [22].

Como se puede observar en la Figura 2.5 se usa un agujero para las cámaras con el fin de tener una imagen menos borrosa ya que se permite el paso punto por punto y se refleja en la imagen invertida mediante líneas de puntos de la figura donde se forma una imagen plana con un centro llamado Centro de Proyección, el cual genera la distancia focal. En la forma de percepción 3D, se define profundidad como la distancia del observador a la superficie; otro concepto es la orientación superficial, se basa en *slant* (inclinación vertical) y *tilt* (dirección de la inclinación) respecto al observador. Se sabe que en tres dimensiones tenemos el ancho, profundidad y la altura del cuerpo; al pasar a una imagen en dos dimensiones se pierden los ángulos y la distancia [22]. Por ejemplo,

las esquinas de las paredes en 3D sabemos que forman ángulos de 90 grados entre los 3 planos, por otro lado, en una imagen 2D se ve como si formara tres ángulos de 120 grados.

De acuerdo a la geometría de la cámara y los parámetros del objeto detectado en la imagen seleccionada por el algoritmo basado en CNN se enviará parámetros característicos tales como orientación, dirección y sentido a un sistema de control de lazo abierto que se encuentra implementado en el sistema de navegación basado en localización y mapeo simultáneo.

2.6.2 Estimación de la posición

Con el uso de la geometría de la cámara y las imágenes captadas por esta, el tamaño del objeto en la imagen presenta información relevante acerca del posicionamiento de este frente a la posición actual de la cámara, es por ello que se buscará centrar la imagen de acuerdo a los requerimientos del proyecto.

Luego de realizar un cuadro delimitador mediante el algoritmo basado en CNN, se trabajará con este para determinar la posición relativa de la persona frente al robot cíclicamente hasta que se logre el posicionamiento correcto.

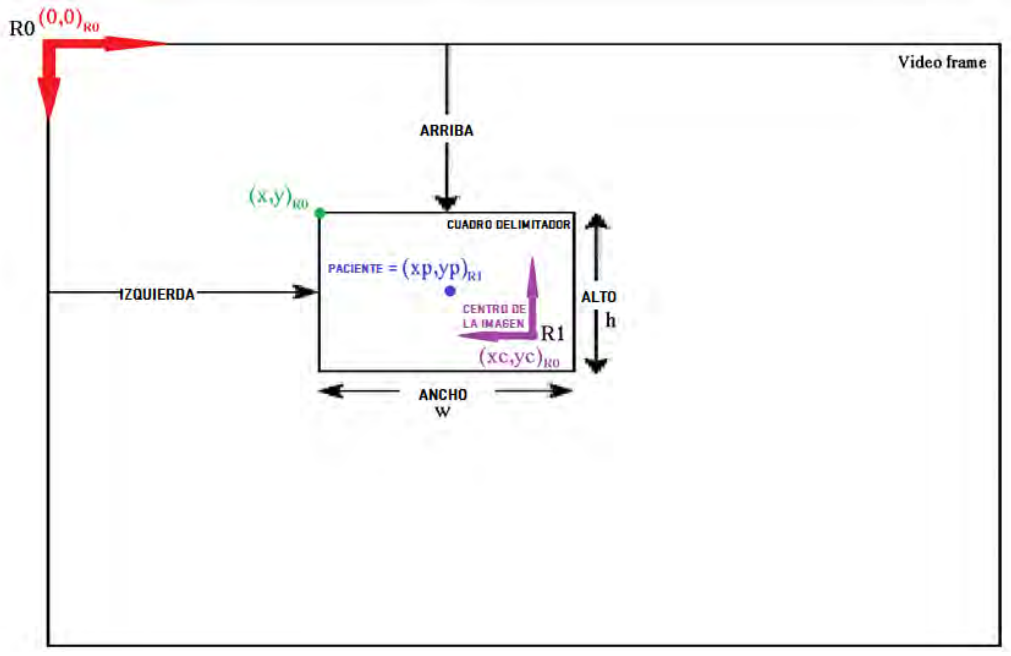


Figura 2.6: Cuadro delimitador dentro de la imagen [32].

Se puede observar en la Figura 2.6 la relación entre el cuadro delimitador y la imagen proporcionada por la cámara, de esta manera usando la información del centro de la imagen respecto a R_0 es posible calcular el centro del objeto detectado dado por R_1, en este caso el paciente, mediante las siguientes ecuaciones:

$$x_p = x_c - \left(x + \frac{w}{2}\right) \quad (2.1)$$

$$y_p = y_c - \left(y + \frac{h}{2}\right) \quad (2.2)$$

Donde x_C e y_c son respectivamente los pixeles 1919 y 1079 que representan el centro de una imagen proporcionada por la resolución de cámara, la cual para esta tesis es de 3840x2160.

Conocidas las coordenadas de píxeles del centro del cuadro delimitador que detecta a la persona, es posible obtener la distancia entre el robot y la persona simplemente extrayendo esta información del píxel correspondiente de la matriz de la cámara de profundidad. La dimensión de esa matriz es igual a la resolución del cuadro RGB adquirido, y para cada posición de píxel, hay un valor en milímetros que representa la distancia de la cámara a lo que ve [32]. De esta manera se determinará los parámetros de movimiento hasta lograr un ancho y largo que pueda asegurar estar a una distancia adecuada de la persona donde se pueda visualizar sin ningún inconveniente.

2.7 MODELO SOLUCIÓN

En la localización de pacientes mediante el procesamiento de imágenes es importante obtener una alta efectividad en el reconocimiento del mismo. Es por ello que se emplea un algoritmo robusto basado en redes neuronales convolucionales para determinar los parámetros de movimiento que tiene que realizar el robot para el posicionamiento respectivo. Es así que en base a lo descrito se ha planteado el modelo que se puede observar en la Figura 2.7.

Figura 2.7: Diagrama de bloques del modelo de solución.

El primer paso, una vez que el robot ingresa a la habitación y usando la cámara e-CAM82_USB, es generar las imágenes con alta resolución para resolver un problema de clasificación según la generación de categorías establecidas. Con esto se determinará si el paciente se encuentra en el ambiente. En esta etapa, el algoritmo basado en redes neuronales convolucionales evaluará la clasificación mencionada anteriormente. El segundo paso es estimar la ubicación relativa “actual” del paciente con respecto al robot mediante parámetros en la imagen de video y geometría de la cámara.

Con esta información, el tercer paso es diseñar una trayectoria deseada que será enviado al algoritmo de control de movimiento del robot a través de su sistema de navegación. Lo anterior según los métodos SLAM presentados en el primer capítulo de esta tesis.

Este sistema de navegación diseñará una ruta para seguir la trayectoria propuesta. No obstante, en cada avance del robot se repetirán los pasos 2 y 3 actualizando las ubicaciones relativas y recalculando la trayectoria deseada, lo cual se repetirá hasta alcanzar el objetivo que es ubicar el robot lo suficientemente cerca del paciente. Cabe indicar que el sistema de navegación del robot cuenta con la instrumentación necesaria para detectar obstáculos que requieran la modificación de la trayectoria planteada. Finalmente, se presenta el siguiente diagrama de flujo para un mayor entendimiento del modelo de solución planteado.

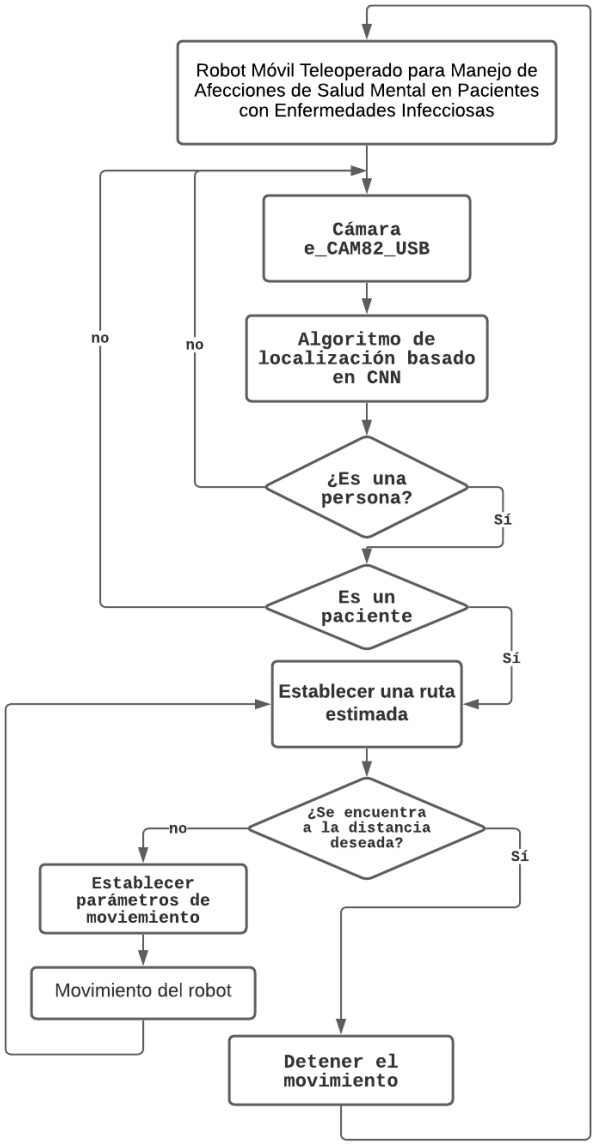


Figura 2.8: Diagrama de flujo del modelo de solución.

CAPÍTULO 3: DISEÑO DE LOS ALGORITMOS

En el presente capítulo, inicialmente se presenta el diseño del diagrama de conexiones de componentes electrónicos del sistema de localización, además se realiza un balance de potencia de consumo del sistema para garantizar un funcionamiento continuo.

Luego se propone el diseño del algoritmo de reconocimiento, así como la selección del algoritmo de estimación de distancia y su respectiva justificación. Finalmente, se realizan los cálculos para hallar las coordenadas basadas en el mapeo simultáneo y la comunicación con el robot.

3.1 ADQUISICIÓN DE LA IMAGEN

Según lo fundamentado en el capítulo anterior, se usará un algoritmo basado en redes neuronales convolucionales para el reconocimiento de personas, de acuerdo a las imágenes adquiridas mediante la e-CAM82_USB, el cual estará implementado en el computador Jetson Nano, que a su vez obtendrá los parámetros de movimiento del robot basados en la estimación de distancia y dirección, los cuales se enviarán al computador central Jetson Xavier NX encargado del sistema de control y navegación.

En la Figura 3.1 se puede observar un esquema de procesos, que inicia con la cámara de modelo e-CAM82_USB y la conexión con el computador Jetson Nano por USB 2.0, que a su vez se encuentra conectado al computador Jetson Xavier NX mediante ROS, el cual es un conjunto de herramientas y softwares para aplicaciones en robótica, debido a que el sistema de navegación y control del robot tiene como computador principal este último. El motivo de la selección de este tipo de comunicación se debe a que el computador principal que realiza la mayor información relevante de la navegación autónoma y la adquisición del mapa generado del entorno en que se encuentra el robot, se encuentra implementado en ROS.

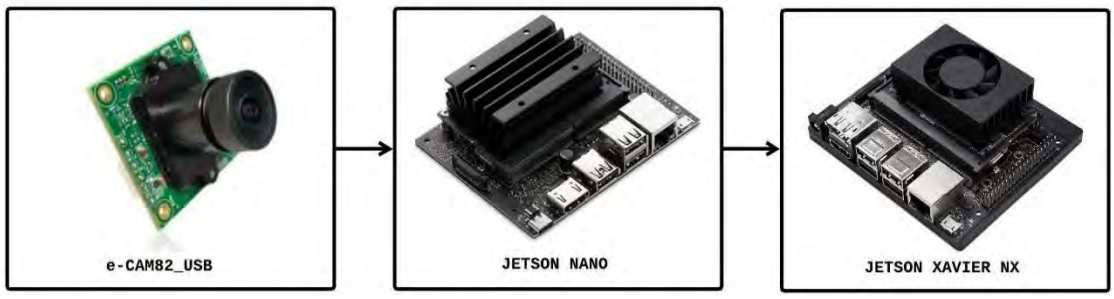


Figura 3.1: Componentes del sistema de localización.

El proceso se basa en la adquisición de las imágenes de una calidad de 8 *megapíxeles*, las cuales serán procesadas mediante el algoritmo de clasificación

escogido, con los resultados de este algoritmo se utilizarán en un modelo de predicción de distancia con el objetivo de generar una ruta estimada en cualquier instante de tiempo, finalmente esta información se enviará al computador central del robot (Jetson Xavier) el cual mediante un lazo de control en lazo cerrado de movimiento del robot establecerá los parámetros de desplazamiento hacia ese punto.

En base a la hoja de datos de los dispositivos se elabora el diagrama de conexiones del sistema de localización propuesto, cabe mencionar que existen otras funciones del proyecto que se realizarán dentro de los computadores pero que no se usarán en el desarrollo del sistema de localización; como por ejemplo, la transmisión en tiempo real de la imagen observada por la cámara, y la teleoperación, es por ello que para fines de esta tesis se muestra el diagrama de conexiones del sistema de localización de pacientes elaborado en el software EasyEda.

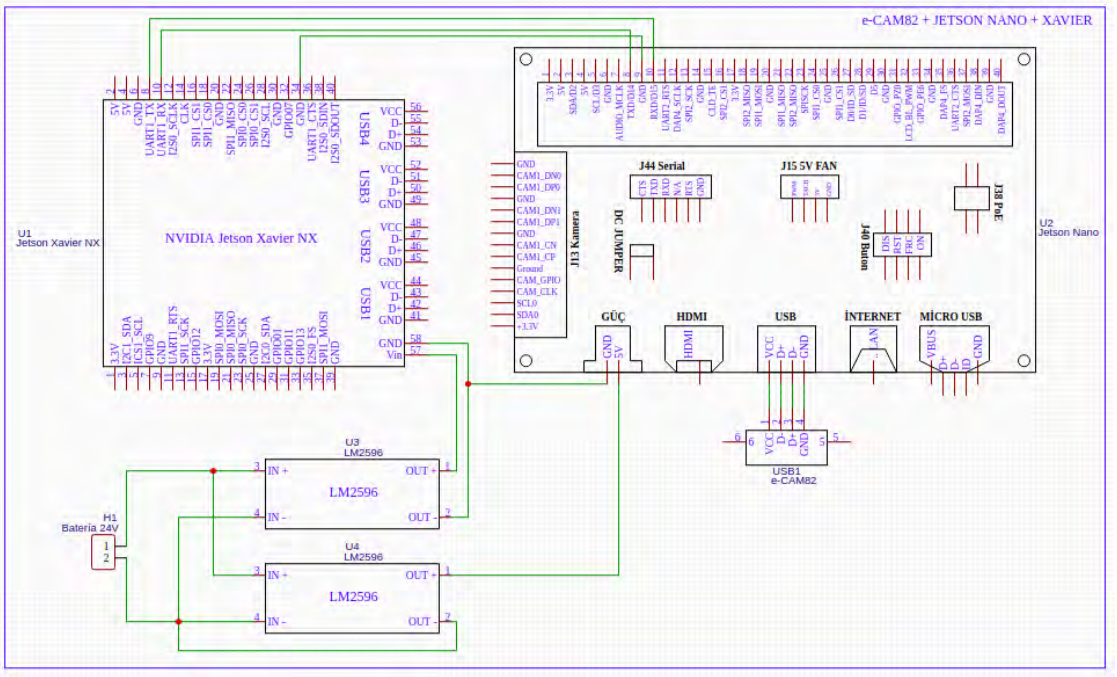


Figura 3.2: Diagrama de conexiones de componentes electrónicos del sistema de localización.

Como se muestra en la Figura 3.2 se usan dos reguladores de tensión a 19 y 5 V para poder alimentar los kits de la Nvidia Jetson Xavier NX y Jetson Nano respectivamente. Por otro lado, la e-CAM82_USB se conecta de manera directa la Jetson Nano por el puerto USB 2.0, alimentándose con 5 V desde la misma conexión.

Basado en los modelos de robots de teleoperación presentados en el primer capítulo y considerando su rango de visión se establece una altura correspondiente a un personal de salud peruano promedio, 1.65 metros, respecto al piso para la altura de la

cámara. Según las especificaciones de la cámara, se muestra la Figura 3.3, el rango de visión del sistema de localización definido por la e-CAM82_USB.

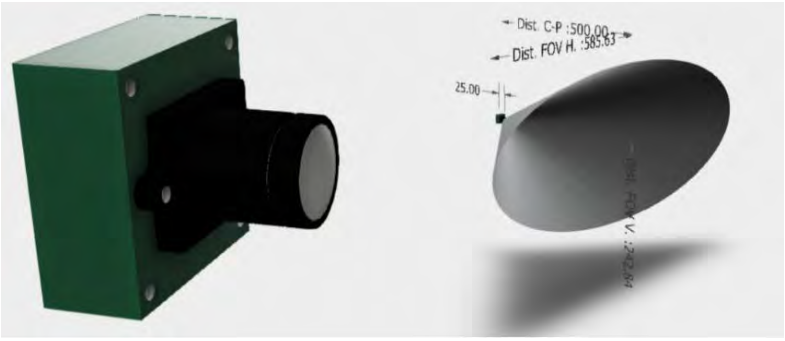


Figura 3.3: Modelamiento de la cámara y su rango de visión.

3.2 BALANCE DE POTENCIA DE CONSUMO DEL SISTEMA

Para iniciar la ejecución del sistema, se debe de alimentar correctamente el computador el cual realizará el algoritmo, por lo que el circuito de alimentación contará con la batería de 24V, el cual se encuentra en el robot móvil, que se conecta al convertidor DC-DC ya que se necesita un cambio de 24V a 5V para su alimentación, adicionalmente a ello, el módulo es capaz de soportar los 4A que requiere la Jetson Nano al trabajar en su máximo rendimiento.

Asimismo, se alimentará la e-CAM82_USB mediante la salida USB del computador Jetson Nano, para no agregar más conexiones a la batería, ya que se usará en otros dispositivos del proyecto. Finalmente, el diagrama de conexiones se muestra en la Figura 3.4.

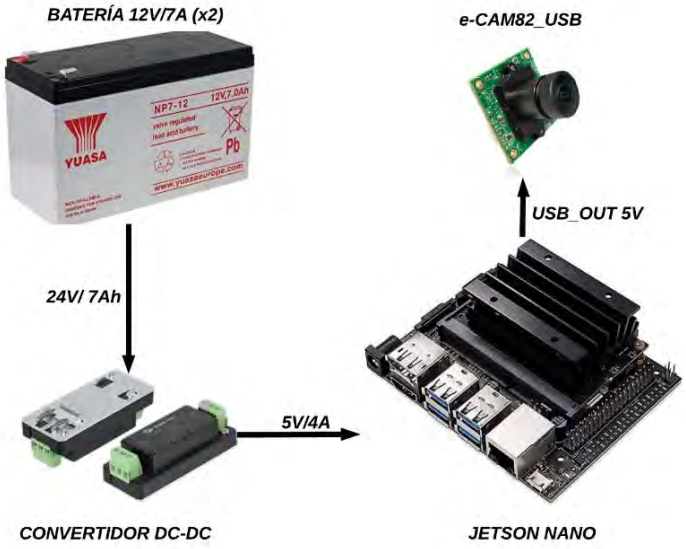


Figura 3.4: Conexiones para la alimentación del sistema.

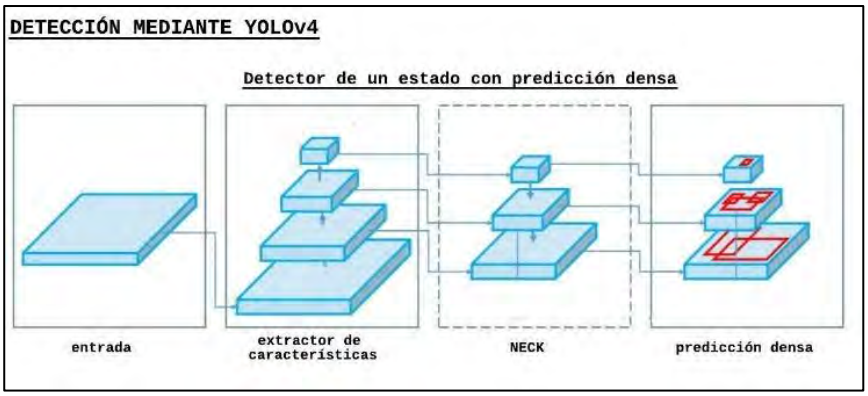
El convertidor DC-DC mostrado en Figura 4.5, es un dispositivo electrónico de la empresa CUI INC escogido por el proyecto ya que ofrece una salida de 5V y una potencia máxima de 20W, lo cual es lo requerido por la Jetson Nano cuando trabaja en su máximo rendimiento. Además, al ser un dispositivo se salida de tensión fija con una eficiencia mayor a 0.9 no es necesario colocar componentes adicionales para controlar esta. El modelo del convertidor DC-DC de 20W de alta eficiencia es PYBE20-DIN.

3.3 IMPLEMENTACIÓN DEL ALGORITMO YOLO v4

Según lo descrito en el capítulo 2 se usará el algoritmo YOLO para el reconocimiento de personas ya que, en comparación con los otros métodos basados en redes neuronales, presenta mejores resultados, de acuerdo a la gráfica de comparación en la Figura 3.5. El algoritmo genera los propios cuadros delimitadores en la imagen de acuerdo a la clasificación generada, como se muestra en la Figura 3.1. Por otro lado, en esta primera etapa se consideran imágenes que permitan extraer los parámetros necesarios para estimar una ruta de acuerdo a la posición de la persona y la camilla donde se encuentra. Además, el algoritmo YOLO v4 se ha desarrollado muy recientemente, utiliza el dataset COCO, el cual representa diferentes situaciones de objetos y personas de manera general con el fin de entrenar al algoritmo para poder clasificar correctamente.

El algoritmo es un clasificador de un estado por predicción densa, es decir, que utiliza el dataset COCO para su entrenamiento, se usa este por su ventaja tanto como otros algoritmos de predicción densa y otros de predicción escasa. En la Figura 3.4 se muestra un resumen de las modificaciones realizadas dentro del *"neck"* partiendo de la base del YOLOv3 para desarrollar la versión que se usará, además de algunas comparaciones con detectores que existen en el mercado de aprendizaje automático.

Este algoritmo es compatible en el entorno de los computadores NVIDIA, para ejecutarlo se necesita las configuraciones del GPU, CUDA y OpenCV dentro de la plataforma. Las modificaciones del mismo algoritmo para el computador Jetson Nano se encuentran en Anexo 1, que en resumen son cambios en la codificación de su archivo "*Make*" con la activación del uso del GPU, configuración para el reconocimiento de la Jetson NANO como un dispositivo de NVIDIA, y el procesamiento mediante el uso de todos sus sistemas de alto rendimiento para una ejecución rápida.



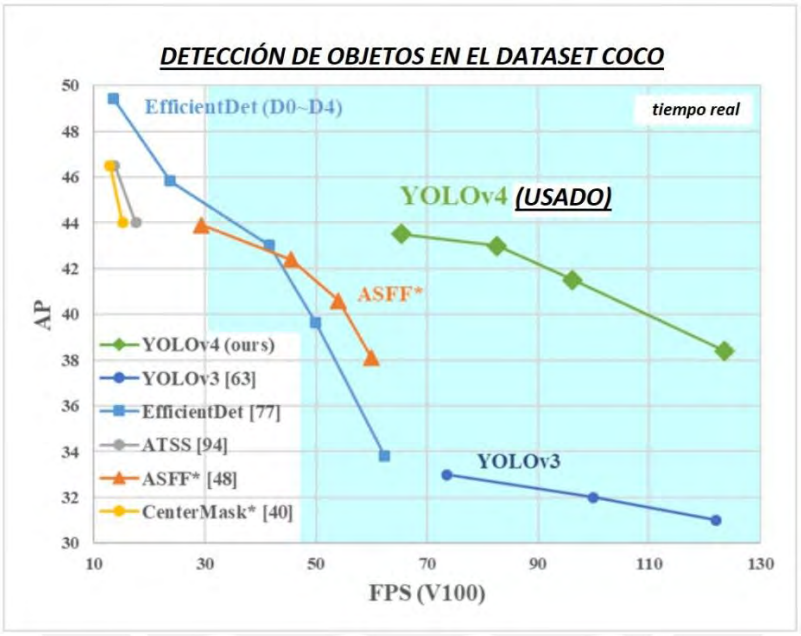


Figura 3.5: YOLOv4 y características principales. [33]

Una vez implementado este algoritmo en el computador, se puede extraer información otorgada como un documento en extensión "*.json", esta extensión es un arreglo de variables de tipo objeto las cuales se extraen para generar el dataset de la orientación y de la distancia. A continuación, se muestra un ejemplo del documento, producto de la operación del algoritmo YOLOv4 implementado en el computador.


```
[
{
 "frame_id":1,
 "filename":"data/lejos.jpeg",
 "objects": [
  {"class_id":62, "name":"tvmonitor", "relative_coordinates":{"center_x":0.275975, "center_y":0.529260,
"width":0.172558, "height":0.196865}, "confidence":0.828425},
  {"class_id":0, "name":"person", "relative_coordinates":{"center_x":0.183088, "center_y":0.777769,
"width":0.143322, "height":0.425009}, "confidence":0.889682}
 ]
}
]
```


Figura 3.6: Salida "*.*json*" del algoritmo YOLOv4.

Como se muestra en la Figura 3.6 , se tiene información de los principales parámetros del cuadro delimitador, como se señaló en el segundo capítulo de la tesis, ofrecida en porcentaje de acuerdo a los pixeles de la imagen, donde “center_x” , “center_y” , “witdh” , “height” se refieren al “centro en x e y”, “ancho” y “alto” del cuadro delimitador respectivamente. A partir de ello se genera un dataset de acuerdo a imágenes existentes adaptadas al desarrollo de esta tesis y generación de data aleatoria que guarda relación con la original puede determinar la orientación la cual se debe mover para centrar la persona.

3.4 EXTRACCIÓN DE LOS PARÁMETROS Y CREACIÓN DEL DATASET

Se diseña el código en Python para extraer los datos del archivos “*.json” ofrecidos por el clasificador, estos datos se usarán para la creación del dataset de aprendizaje, dicho de otro modo, se tendrá un archivo “*.xlsx”. Para el entrenamiento de la red se toman las imágenes con una resolución de 0.5 metros en la estimación de distancia, para la cual los siguientes parámetros:

- Distancia en x al centro de la “persona”.
- Distancia en y al centro de la “persona”.
- Área del cuadro delimitador.
- Ancho del cuadro delimitador.
- Largo del cuadro delimitador.

Si bien, el área del cuadro delimitador está correlacionada con el ancho y largo de este, no siempre engloba a la persona en su totalidad, dado que al ser un paciente echado en camilla puede detectar desde la mitad del cuerpo o inclusive, en algunos casos, solo la cabeza del mismo. Es decir, en el caso que se detecte solamente la cabeza del paciente y esta área sea similar a una con tronco completo, se puede generar una distancia errónea, por ello se considerarán los siguientes parámetros adicionales:

- Área de la “camilla”.
- Área de la “camilla” respecto a la “persona”.

Por otro lado, para la estimación de la orientación con una resolución de 15°, para un ángulo de visión de 110° que ofrece la cámara, son 7 los intervalos que se clasificarán de acuerdo a los siguientes parámetros:

- Distancia en x al centro de la “persona”.
- Distancia en x al centro de la “camilla”.

En la Figura 3.7 se muestra el módulo utilizado para la adquisición de parámetros, que consta de dos partes principales: la cámara e-CAM82_USB a 1.65 metros de distancia respecto al piso, conectada al computador Jetson Nano, con el fin de simular de la ubicación final de la cámara en el robot.

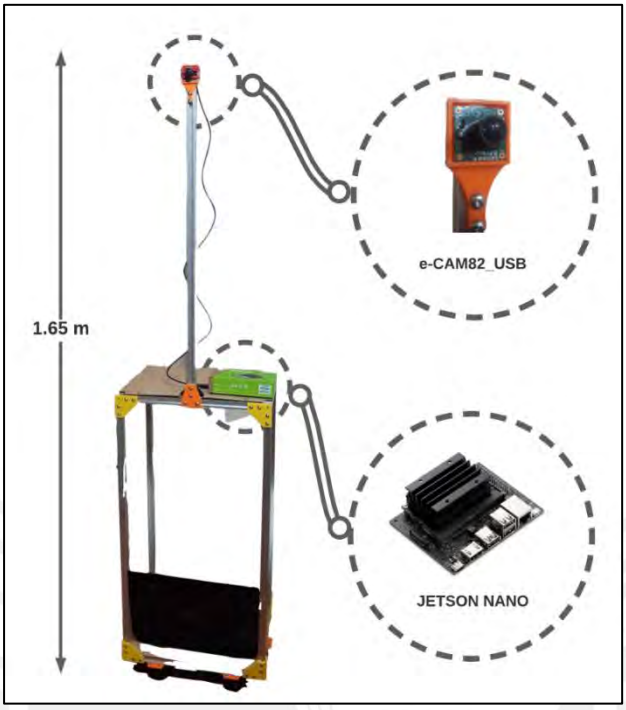


Figura 3.7: Módulo fabricado para la toma de parámetros y creación del dataset.

Si bien, se recomienda contar con miles de imágenes recolectadas para tener un mejor entrenamiento, dado el contexto actual, se tomarán en principio decenas de muestras reales de donde a partir de ellas se generará data aleatoria, posteriormente para cuando el robot se encuentre implementado se ha diseñado la función en Python con el fin de agregar estas nuevas experiencias al archivo "*.xlsx" y generar un nuevo modelo predictivo, a partir de las nuevas experiencias obtenidas, para lograr un aprendizaje automático.

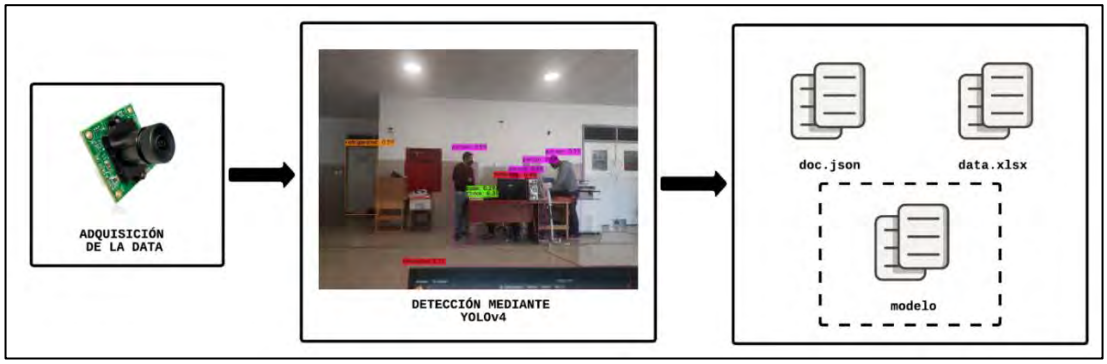


Figura 3.8: Diagrama de generación del modelo predictivo.

Creación del dataset para estimación de dirección: Como se explicó anteriormente, se tiene 7 intervalos para una resolución de 15° de tal forma que se asigna una salida específica a cada intervalo mostrada en la Tabla 1.4.

Tabla 3.1. Descripción del dataset del estimador de dirección.

Intervalo	Descripción	Target
[-55°;-40°]	Rotar en sentido antihorario 47.5°	1
[-40°;-25°]	Rotar en sentido antihorario 32.5°	2
[-25°;-10°]	Rotar en sentido antihorario 17.5°	3
[-10°;10°]	No Rotar	4
[10°;25°]	Rotar en sentido horario 17.5°	5
[25°;40°]	Rotar en sentido horario 32.5°	6
[40°;55°]	Rotar en sentido horario 47.5°	7

Creación del dataset para estimación de distancia: Para esta aplicación, se considera las condiciones indicadas en el capítulo 1 de la tesis: la sala contará con una camilla, la cámara se encontrará en una posición fija, la distancia máxima es 5 metros del paciente al robot y el paciente se considera permanentemente en la camilla de recuperación. Cumpliendo las condiciones indicadas, se asigna una salida específica mostrada en la Tabla 1.5.

Tabla 3.2. Descripción del dataset del estimador de distancia.

Intervalo	Target	Intervalo	Target
[0.0 ; 0.5] m	1	**[2.5 ; 3.0] m**	6
[0.5 ; 1.0] m	2	**[3.0 ; 3.5] m**	7
[1.0 ; 1.5] m	3	**[3.5 ; 4.0] m**	8
[1.5 ; 2.0] m	4	**[4.0 ; 4.5] m**	9
[2.0 ; 2.5] m	5	**[4.5 ; 5.0] m**	10

Consideraciones importantes del archivo "xlsx": Para la creación de estos archivos se tienen los siguientes parámetros:

3.4.1 Estimación de dirección

Área del cuadro delimitador (*tamaño*), posición del punto inferior izquierdo del cuadro delimitador (*x1*, *y1*), posición del punto superior derecho del cuadro delimitador (*x2*, *y2*), centro del objeto detectado en el eje "x" (*centrox*), centro del objeto detectado en el eje "y" (*centroy*), centro de la resolución de la imagen (*centrox_img* = 1919; *centroy_img* = 1079), diferencia del centro con el objeto detectado en el eje "x" (*difx*) y en el eje "y" (*dify*) y por último la salida o etiqueta que se le asignará (*target* = [1; 7]).

3.4.2 Estimación de distancia

Área del cuadro delimitador de la persona (*tamaño*), posición del punto inferior izquierdo del cuadro delimitador de la persona (*x1*, *y1*), posición del punto superior derecho de la persona (*x2*, *y2*), posición del punto inferior izquierdo de la camilla (*x3*, *y3*), posición del punto superior derecho del cuadro delimitador de la camilla (*x4*, *y4*), centro de la persona detectada en el eje "x" (*centropx*), centro de la persona detectada en el eje "y" (*centropy*), centro de la camilla detectada en el eje "x" (*centrocx*), centro de la camilla detectada en el eje "y" (*centrocy*), centro de la resolución de la imagen (*centrox_img* = 1919; *centroy_img* = 1079), diferencia del centro con el objeto detectado en el eje "x" (*difx*) y en el eje "y" (*dify*) y finalmente la salida que se le asignará (*target* = [1; 10]).

3.5 DESARROLLO DEL SISTEMA

Tal como se presenta al inicio de este capítulo, esta etapa de diseño se divide en desarrollo del sistema para predicción de dirección y predicción de distancia. Para el primer caso se muestra el diagrama de flujo en la Figura 3.9.

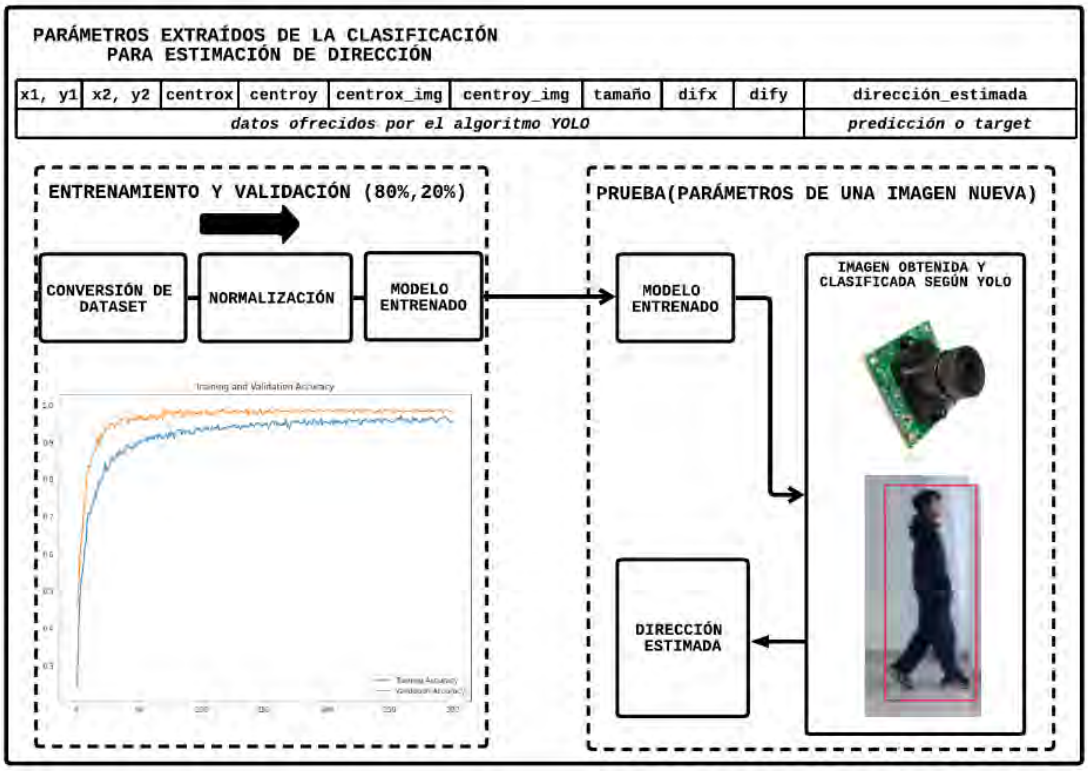


Figura 3.9: Diagrama de bloques del estimador de dirección.

Según la Figura 3.9 se genera un modelo, el cual es realizado mediante redes neuronales, al ser todas las variables de carácter numéricas, se obtiene el diagrama de red neuronal en la Figura 3.10, que representa el modelo de predicción de orientación.

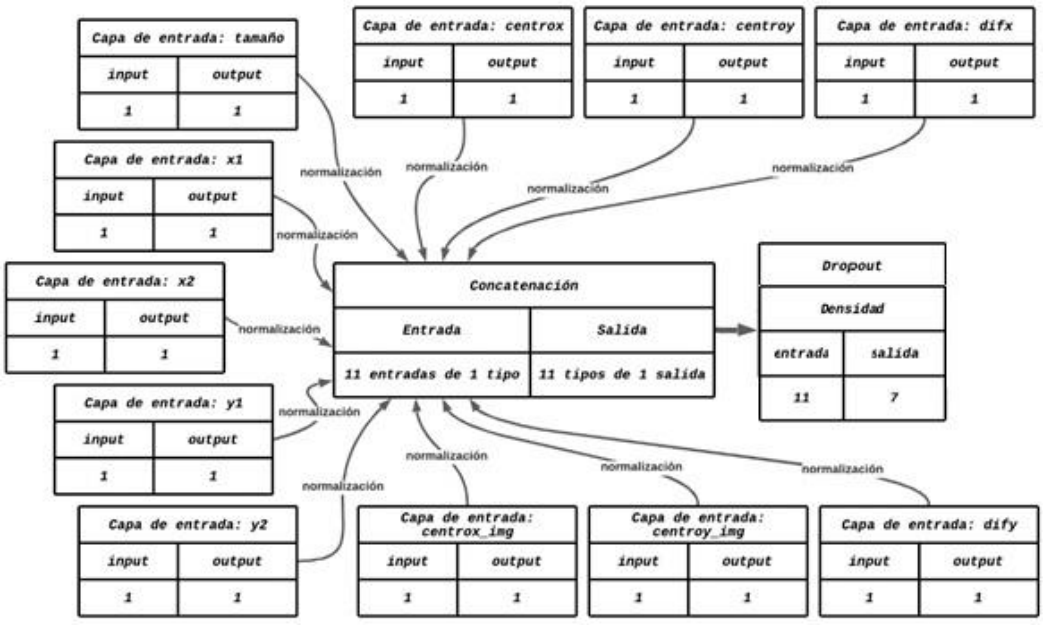


Figura 3.10: Diagrama de la red neuronal del estimador de dirección.

Asimismo, para el segundo caso se muestra el diagrama de flujo en la Figura 3.10. Se puede observar que, a diferencia del anterior algoritmo, tiene más parámetros.

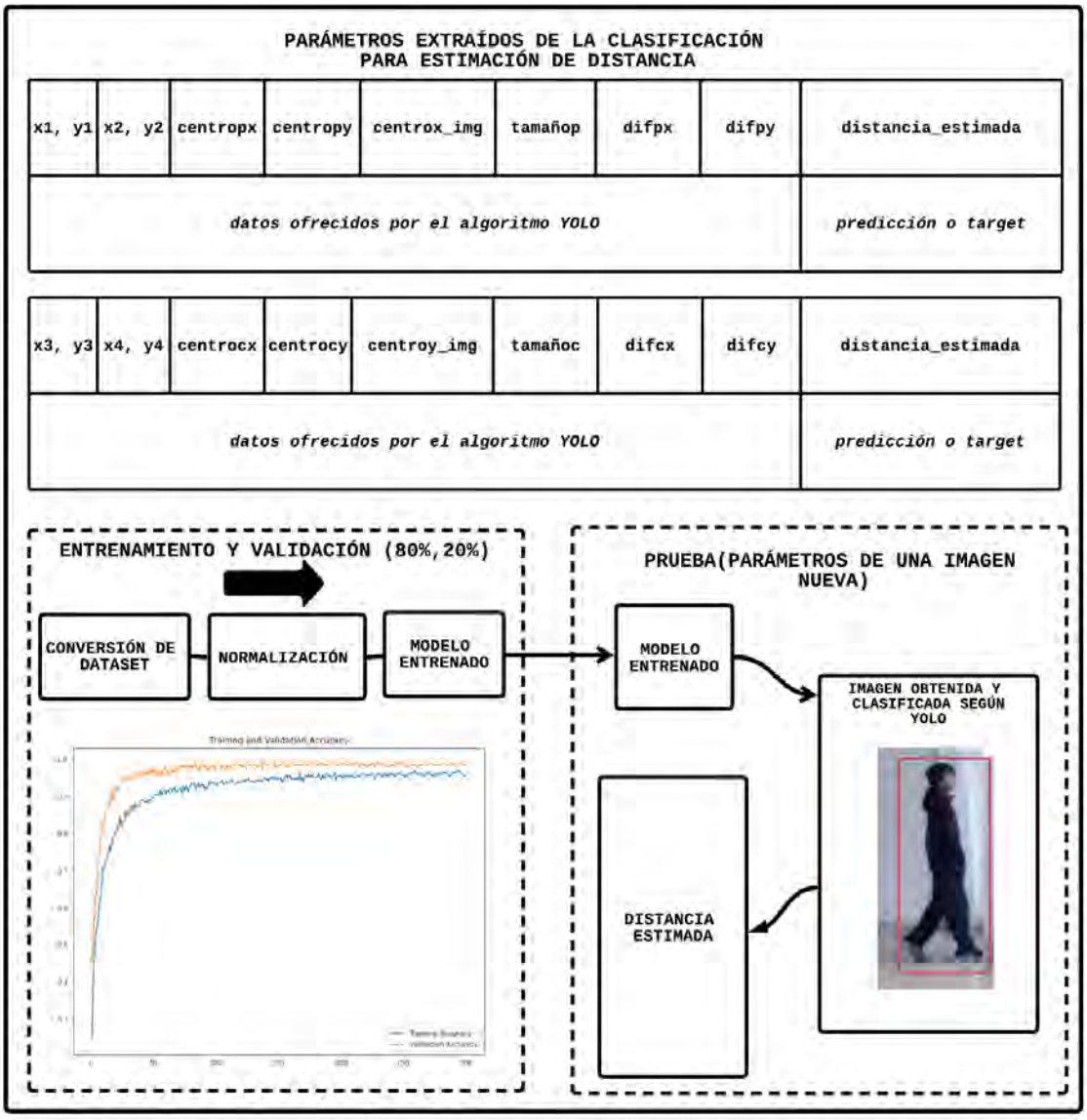


Figura 3.11: Diagrama de bloques del estimador de distancia.

Tanto en las Figuras 3.9 y 3.11 muestra la división en la data de entrenamiento y validación como 80 y 20 % respectivamente del total de data generada, de las cuales mediante la normalización de las entradas se genera un modelo entrenado, el cual posteriormente se extraerá y se guarda como función, de tal manera que para las aplicaciones de otras funciones del robot bastará colocar los parámetros requeridos en el modelo y generará la predicción, tanto como la orientación y la distancia. Adicionalmente, dentro del algoritmo de estimación de distancia se consideran los casos de la “no detección”, tanto de: la camilla, la persona o ambas; para ello, se prevalece el área de la camilla frente a la persona en caso que se detecten ambos porque al detectar

a la persona puede ser una fracción de su cuerpo como el rostro, y generaría una predicción errónea, para los otros casos se tomará el área de la persona.

Por otro lado, estas operaciones de predicciones se realizan de manera continua en el instante que se termina la predicción, se obtiene la imagen de video desde otra función de transmisión realizada por el robot, para evitar obstruirse en un lazo de una función secundaria, esta frecuencia de operación depende de las aplicaciones y requerimientos del proyecto. Finalmente, según la Figura 3.11 se genera un modelo predictivo, el cual es realizado mediante redes neuronales, al ser todas las variables de carácter numéricas, se obtiene el diagrama de red neuronal en la Figura 3.12, que representa el modelo de predicción de distancia.

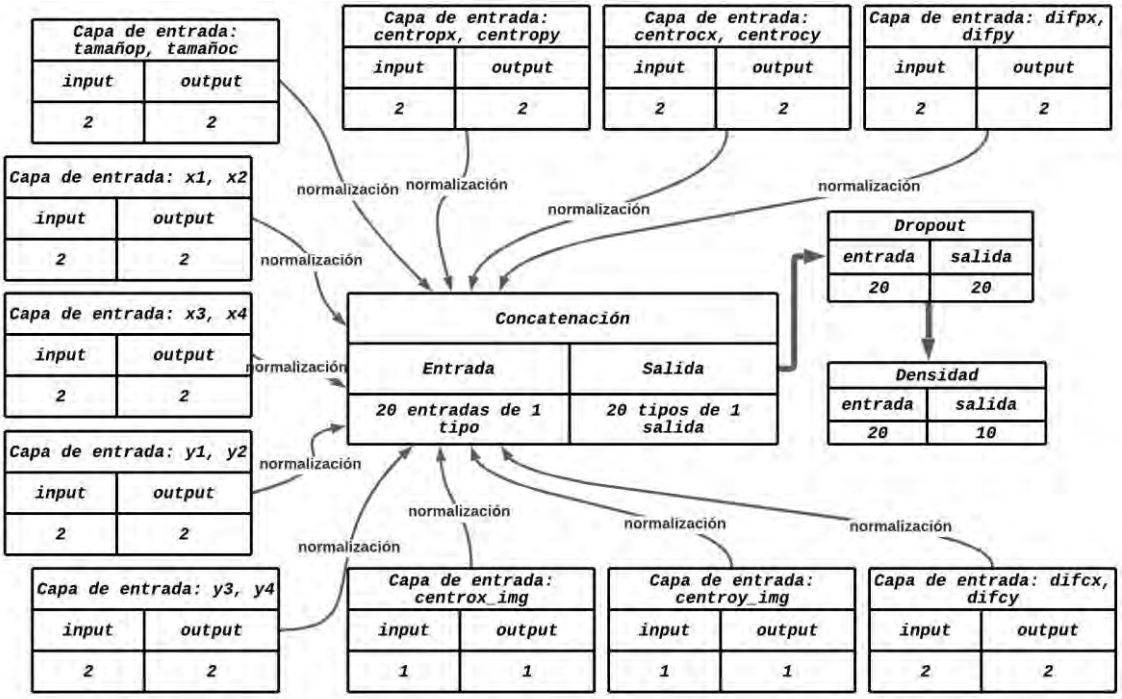


Figura 3.12: Diagrama de la red neuronal del estimador de distancia.

Adicionalmente a la creación de ambos modelo, se aplicará una aprendizaje de los mismos puesto que generalmente el primer modelo generado tiene una precisión no deseable, es por ello que se consideran los conceptos de *batch* y *epoch*,. El primero se refiere a la cantidad de ejemplos que se harán en un ciclo de entrenamiento; mientras que el segundo indica la cantidad de ciclos de entrenamientos que se harán la cantidad de ejemplos seleccionados. Para el diseño del sistema de localización se usa un *batch* de 32 y 300 de *epochs*.

Finalmente, se considera la recolección de la data según la experiencia del robot durante el proceso de acercamiento hacia la persona. Estos parámetros se agregarán

al dataset ".xlsx" con su respectivo target y después de haber realizado todas las funciones necesarias del robot se procede a generar un nuevo modelo, el cual contiene los nuevos parámetros y validaciones, de esta mana se tiene un sistema de estimación de la persona o paciente con aprendizaje automático según las aplicaciones realizadas.

3.6 METODOLOGÍA PARA LA VALIDACIÓN

Para el análisis de la evaluación del resultado del algoritmo se emplean las métricas de clasificación proporcionadas por la librería "*sklearn*". Para las siguientes métricas se define "*precision*" como la cantidad de predicciones relevantes dentro de los elementos seleccionados como predicciones correctas, y se representa mediante la razón de los verdaderos positivos y los elementos positivos. Asimismo, se define "*recall*" como la cantidad de predicciones relevantes dentro de los elementos seleccionados correctamente ya sea verdadera o falsa, y se representa mediante la razón de los verdaderos positivos y la suma de los elementos falsos negativos con estos.

- ***Precision*** $$P = \frac{TP}{TP+FP} \quad (3.1)$$
- ***Recall*** $$R = \frac{TP}{TP+FN} \quad (3.2)$$

Además, estos parámetros se representan en curvas representativas de acuerdo a las métricas obtenidas, es por ello que, para la validación de la predicción del sistema diseñado, se tendrá en cuanta las siguientes curvas:

Curva de la precisión del entrenamiento (*training accuracy*): Se muestra la precisión del entrenamiento a lo largo de la cantidad de ciclos o *epochs*.

Curva de la precisión de la pérdida (training *loss*): Se muestra la pérdida del entrenamiento a lo largo de la cantidad de ciclos o *epochs*.

Curva de la precisión del entrenamiento (*validation accuracy*): Se muestra la precisión de la validación a lo largo de la cantidad de ciclos o *epochs.*

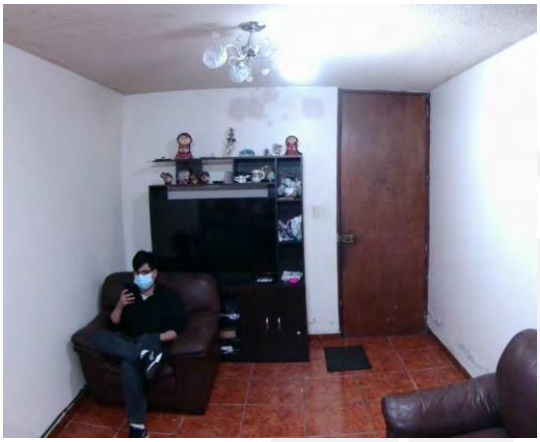
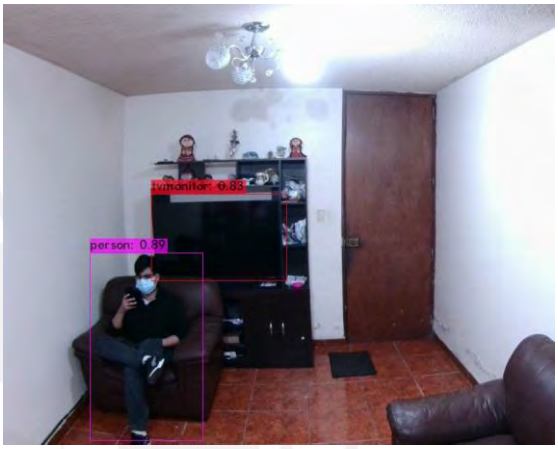
Curva de la precisión de la pérdida (*validation loss*): Se muestra la pérdida del entrenamiento a lo largo de la cantidad de ciclos o *epochs.*

3.7 Descripción de la aplicación del sistema de localización

Se describe toda la operación del sistema mediante el módulo fabricado mostrado en la Figura 3.6, además de la adquisición de las imágenes por la e-CAM82_USB y procesadas en el computador Jetson NANO de la siguiente manera: Primero el robot posicionado en la entrada de la sala de recuperación, podrá localizar a la persona mediante el algoritmo YOLOv4 en primera instancia o en cualquier otro caso a la

persona y a la camilla, si bien se establece una ruta para este primer análisis esta cambiará de acuerdo a las imágenes adquiridas posteriormente. Se muestra en la Tabla 3.3 el estado inicial del módulo elaborado el cual simula al robot, esta imagen adquirida de video se procesa mediante en algoritmo YOLOv4 para generar el archivo "*.json" con la siguiente descripción mostrada en la misma tabla.

Tabla 3.3. Descripción de la aplicación del sistema. Parte1.

Imagen adquirida	**Imagen procesada**
	

Archivo "*.json" generado por el clasificador


```
[
{
 "frame_id":1,
 "filename":"data/lejos.jpeg",
 "objects": [
  {"class_id":62, "name":"tvmonitor", "relative_coordinates":{"center_x":0.275975, "center_y":0.529260,
"width":0.172558, "height":0.196865}, "confidence":0.828425},
  {"class_id":0, "name":"person", "relative_coordinates":{"center_x":0.183088, "center_y":0.777769,
"width":0.143322, "height":0.425009}, "confidence":0.889682}
 ]
}
]
```


De estos parámetros se extraen las variables de acuerdo a cada red neuronal diseñada, para la orientación se muestras las entradas y salidas generadas por el algoritmo en la Figura 3.13. De la misma forma, se muestran para la estimación de distancia.

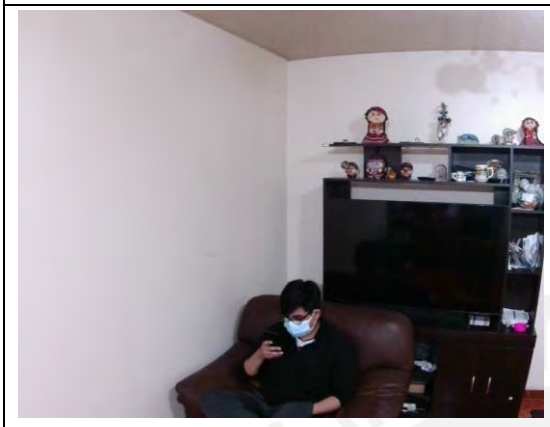
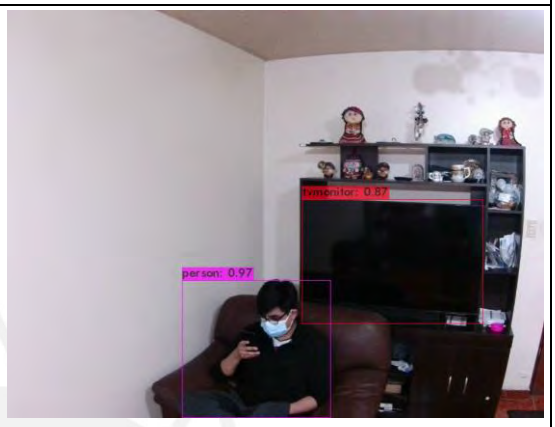
```
}

input_dict = {name: tf.convert_to_tensor([value]) for name, value in sample.items()}
predictions = model.predict(input_dict)

print(
   predictions.round(2)
)
```

```
[[0. 1. 0. 0. 0. 0. 0.]]
```

Figura 3.13: Ejemplo de estimación en la salida generada por el algoritmo.

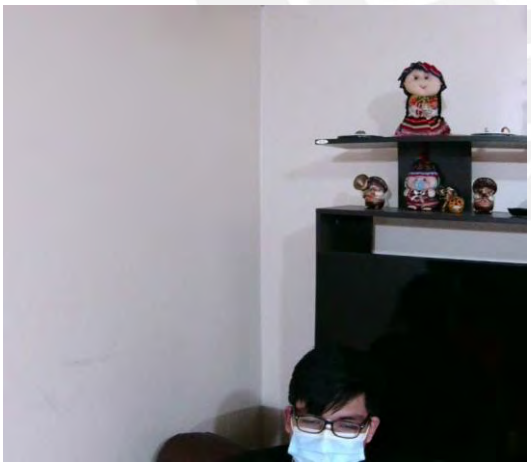
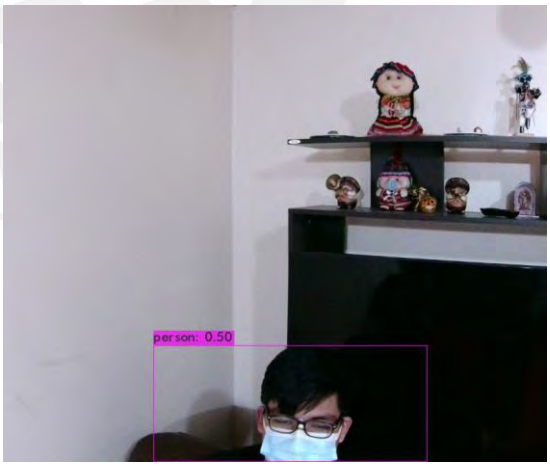
De los resultados se interpreta la siguiente descripción**: "El robot debe rotar 32.5° en sentido antihorario y avanzar porque se encuentra a 3.5 metros de distancia"**, luego de un periodo se extrae la siguiente imagen y se repite la operación anterior.

Tabla 3.4. Descripción de la aplicación del sistema. Parte2.

Imagen adquirida	Imagen procesada
	

Entonces, se genera otro archivo "*.json" con la descripción, entradas y salida generadas después de la aplicación en modelos entrenados, de los resultados obtenidos se interpreta la siguiente descripción**: "El robot no debe rotar pero sí avanzar hacia adelante porque se encuentra a 2 metros de distancia"**, luego de un periodo se extrae la siguiente imagen y se repite la operación anterior.

Tabla 3.5. Descripción de la aplicación del sistema. Parte3.

Imagen adquirida	Imagen procesada
	

Finalmente, se adquiere un último archivo "*.json" con la descripción, entradas y salida generadas después de la aplicación en modelos entrenados, de los resultados obtenidos se interpreta la siguiente descripción: **"El robot no debe rotar ni avanzar porque ya se encuentra a 0.5 metros de distancia"**, y se procede a funciones propias del proyecto, tales como una videoconferencia.

CAPÍTULO 4: PRUEBAS EXPERIMENTALES Y RESULTADOS

En este último capítulo se presentan las pruebas experimentales desarrolladas en las habitaciones que a la fecha de este trabajo vienen siendo equipadas para pacientes que se encuentran en cuidados intensivos(UCI) en el Hospital Regional de Cañete. Si bien las habitaciones cuentan con el equipamiento real en relación a camas, equipo biomédico, entre otros, los ambientes no están todavía ocupados por pacientes, los cuales fueron simulados para este estudio. Sin embargo, las condiciones reales de espacios, alturas y distancias proporcionaron datos muy relevantes para la validación de los trabajos de la presente tesis.

En este capítulo se presentan resultados de la aplicación de los algoritmos diseñados, así como la simulación en software para la validación y comprobación de la comunicación en ROS (“Robot Operating System” por sus siglas en inglés). Esta comunicación es esencial para el enlace entre el sistema desarrollado en esta tesis y el sistema de navegación del robot que controla sus actuadores para su desplazamiento.

Finalmente, se muestran los resultados publicados, los cuales referencian las coordenadas del paciente dentro del mapa generado por el robot.

4.1 POSICIONAMIENTO DEL ROBOT EN LA PUERTA DE LA HABITACIÓN

Inicialmente el sistema de navegación del robot basado en un sistema LIDAR generó un mapa del piso donde se desplazará el robot y podrá luego ser dirigido de manera automática. En la Figura 4.1 y 4.2 se muestra el mapa del piso de pacientes autorizado para ensayos y el mapa generado por el robot respectivamente.

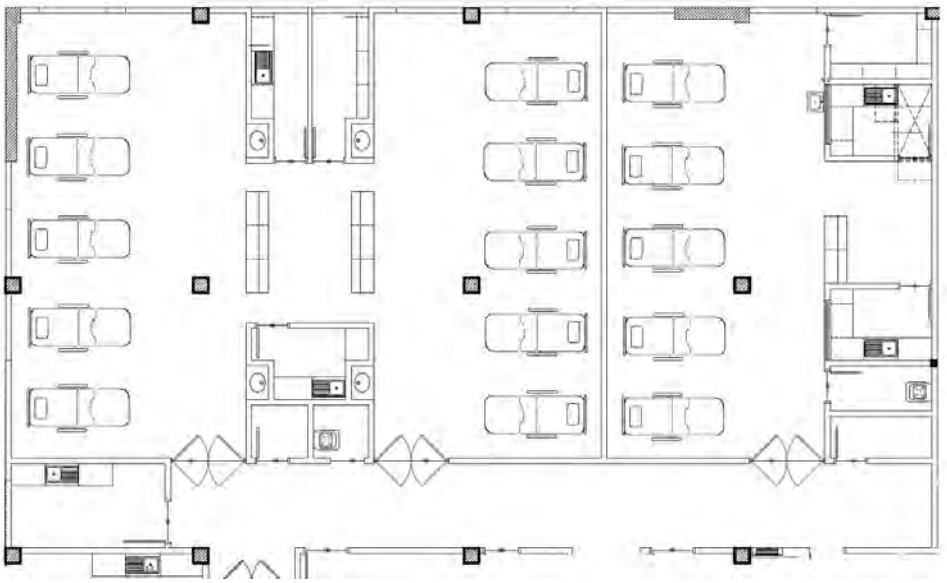


Figura 4.1: Plano de piso donde se realizaron las pruebas Hospital Regional de Cañete.

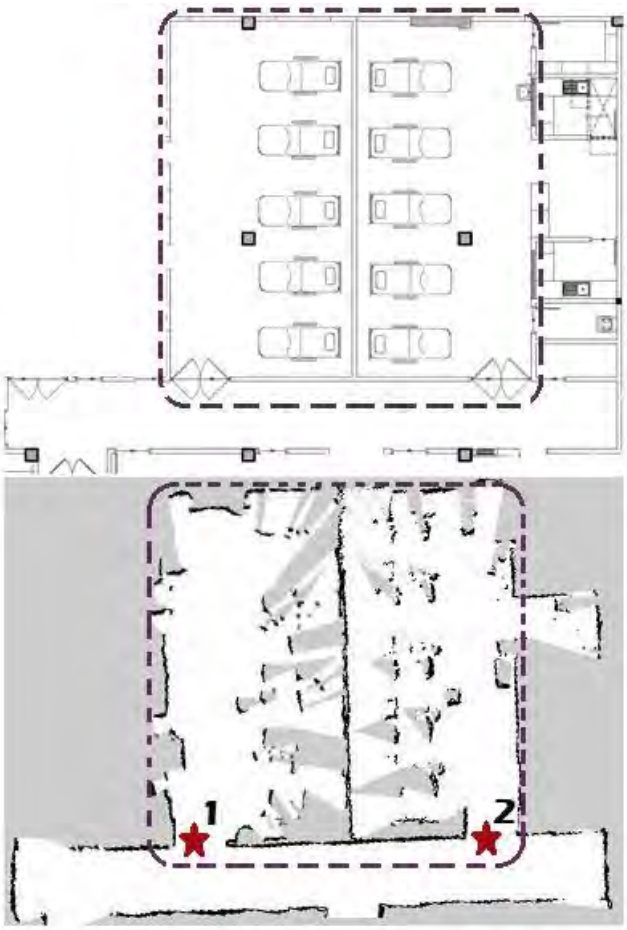


Figura 4.2: Mapa generado por el robot y ubicación inicial para las pruebas.

En la Figura 4.2 se muestra la comparación entre el mapa generado por el sistema y el mapa real, también se muestran los dos puntos de partida en los que inicialmente se posiciona al robot para las pruebas de localización del paciente. El sistema de navegación del robot puede movilizar al robot dentro del plano desarrollado, luego de ingresar a la habitación se hace uso del sistema de localización y ubicación desarrollado en la presente tesis.

4.2 SISTEMA DE LOCALIZACIÓN

Según lo fundamentado en el capítulo anterior, se usará un dataset creado con valores reales y aleatorios almacenados en la Jetson Nano, además de los modelos pre entrenados de distancia y orientación. Posteriormente se publicará en el formato específico de cuatro valores de tal manera que se envíen una “pose” como lo requiere el computador Jetson Xavier NX, debido a que las coordenadas enviadas para el movimiento dentro del sistema de navegación son mediante cuatro parámetros: distancia en el eje “x”, distancia en el eje “y”, orientación y la distancia del objeto respecto al piso, teniendo como centro de coordenadas la posición del robot dentro del mapa creado por el mismo. Finalmente, mediante las configuraciones de modo esclavo y

maestro en los nodos de la Jetson Nano y la Jetson Xavier NX respectivamente, se lee el tópico con la información enviada en forma de pose. En la Figura 4.3 se muestra el diagrama de bloques del sistema desarrollado. En el Anexo 1 se muestra la codificación respectiva de la e-CAM82_USB para la adquisición de datos por la Jetson Nano.

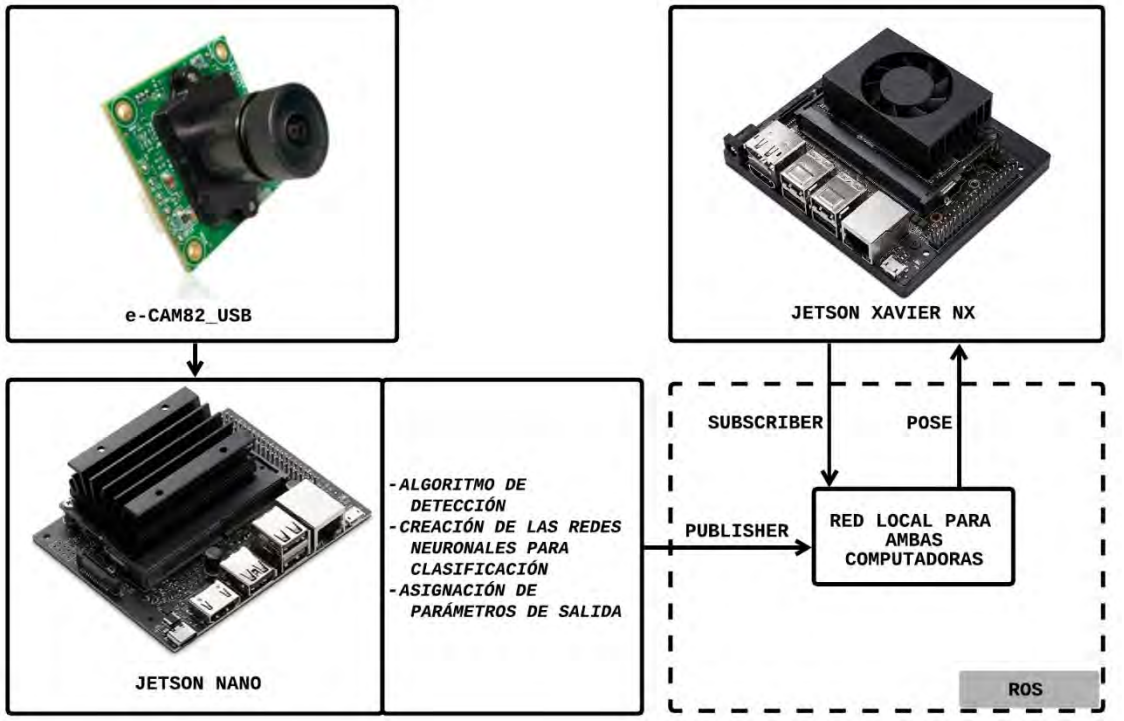


Figura 4.3: Diagrama de conexiones del sistema de localización.

4.3 IMPLEMENTACIÓN DEL ALGORITMO EN EL SISTEMA

Figura 4.4: Diagrama de bloques del sistema de localización.

Como se muestra en la Figura 4.4, el algoritmo se divide en cuatro funciones principales las cuales se pueden visualizar en el Anexo 2. Se inicia el algoritmo mediante la función *asig_model*, la cual asigna el modelo pre entrenado a una variable dentro del algoritmo, en este caso las variables son modelo y modelo2 los cuales representan el modelo de predicción de orientación y de distancia respectivamente; en formato de red neuronal de *Keras* de la librería *Tensorflow* con extensión .h5 desarrollado en el software de Google Colaboratory. Segundo, la función *red_neuronal* de la misma manera que la primera función hace la lectura de un archivo cargado previamente a la ejecución del algoritmo, en este caso es la generación del cuadro delimitador generado por YOLO en formato "*.json*" dentro del Jetson Nano; esta función genera las predicciones tanto de la orientación como el tamaño. Tercero, la función *asig_out* tiene como entrada las predicciones generadas para poder determinar los parámetros de pose específicas para poder publicarlas en un tópico determinado. Finalmente, se publica en un nodo esclavo dentro de una red local para poder leerlas por un suscriptor en la Jetson Xavier NX dentro de la misma red local.

Todo el proceso de generación de pose en un formato establecido es por el uso de ROS para la comunicación con la Jetson Xavier de tal manera que este se ejecute como maestro recibiendo información del nodo esclavo en la Jetson Nano.

4.4 VALIDACIÓN DEL SISTEMA DE LOCALIZACIÓN EN EL HOSPITAL REZOLA DE CAÑETE

En esta sección se presentan los resultados de los trabajos de validación experimental realizados en habitaciones del Hospital Rezola de Cañete de acuerdo al proceso explicado de adquisición de data, generación del cuadro delimitador, establecimiento de parámetros y envío. Respecto a la simulación en una sala de recuperación de hospital, se pudo obtener una habitación con una camilla en el interior, dentro de este entorno se parte desde la puerta de la habitación como punto inicial del robot(Figura 4.2) y se ejecuta el algoritmo. En la Figura 4.5 se muestra la primera secuencia de imágenes obtenida, con su respectivo archivo generado con la información del objeto detectado.

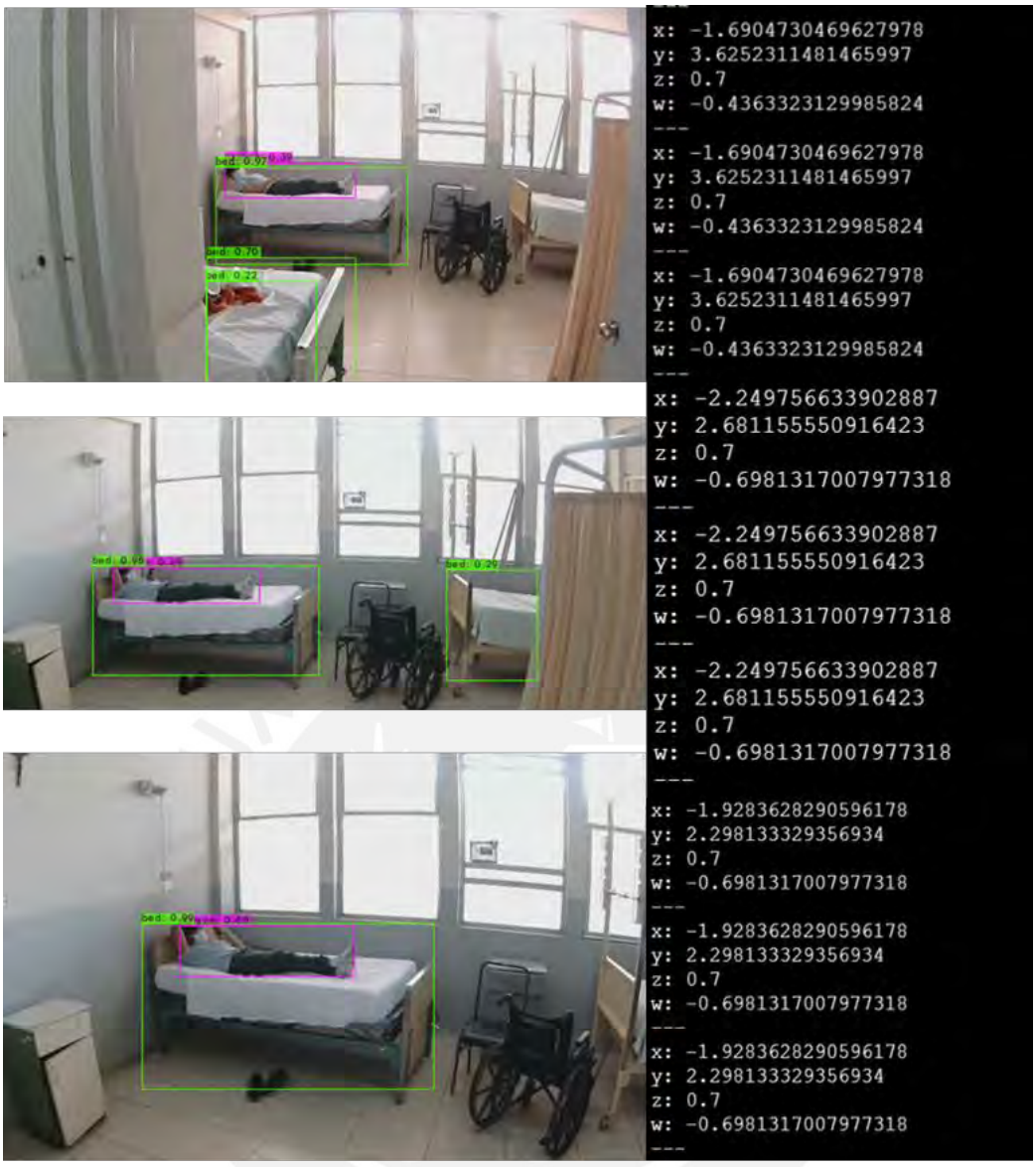


Figura 4.5: Primera secuencia de imagenes adquiridas por el sistema.

Como se puede apreciar en la primera posición, se han detectado las variables de cama y persona mediante el sistema basado en YOLO, esta imagen es la referencia del archivo .json que tiene la información relevante que se utiliza como entrada del sistema de localización. Por otro lado, la pose en un espacio libre en ROS que se requiere consta de dos partes: la posición y la orientación, el primero se genera en formato de “Point” y el segundo en “Quaternion” para su respectivo envío en el tópico.

Una vez obtenidos los datos de orientación y distancia se derivan a los parámetros de pose, los cuales se modifican para enviarlos en traslación y orientación; el primero de estos simplemente es en formato (x,y,z) tomando como origen de coordenadas la base del robot mientras que el segundo (x,y,z,w) al ser el robot móvil invariante en rotación tomando como eje “x” e “y”, solo se varia los parámetros z y w, de acuerdo al

ángulo generado por su dirección al objeto detectado. Es por ello que en los resultados se presentan los parámetros de x , y , z que corresponden al punto de traslación y w a la orientación generada por la dirección al objeto.

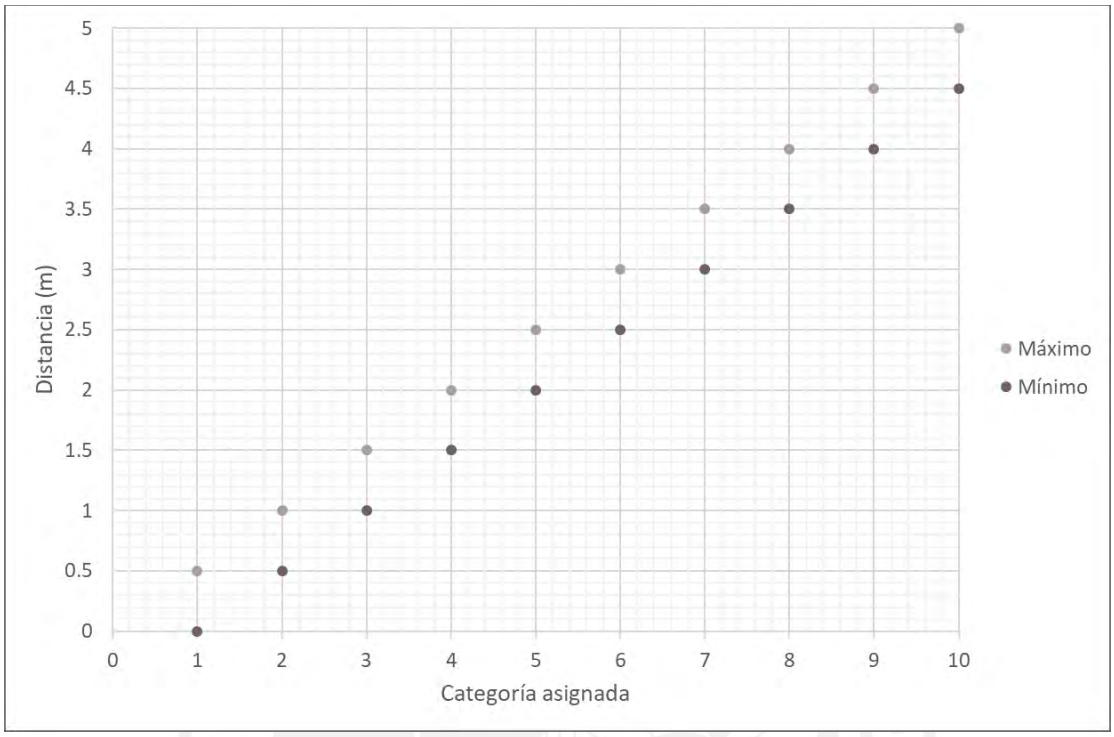


Figura 4.6: Máximos y mínimos de detección por categoría .

Asimismo, se muestra la primera lectura en el tópico publicado una vez ejecutado el algoritmo diseñado. El sistema esta discretizado de tal manera que los valores que aparecen en la visualización del tópico son referenciales (la raíz de la suma de cuadrados de los ejes x e y es el valor de la categoría, asignada de acuerdo a la Figura 4.6, es por ello que se repiten mientras se encuentran dentro del rango colocado en el intervalos, los cuales se muestran en la Tabla 3.1 y 3.2 para los valores de orientación y distancia respectivos.

Luego de ejecutarse el movimiento correspondiente, un giro antihorario de 32.5° y avanzar aproximadamente medio metro se toma la detección nuevamente y se procede a analizar los resultados generados. También se muestra la segunda posición en la Figura 4.6, con su respectivo archivo generado con la información del objeto detectado. Asimismo, al igual que la primera adquisición se muestra la detección de las variables de cama y persona por la ejecución del YOLO; de la misma forma que la primera pose, se realiza el envío de parámetros de acuerdo a la discretización realizadas en las Tablas 3.1 y 3.2 al igual que la primera publicación realizada en el tópico. Se gira el módulo 17.5° en sentido antihorario y avanza aproximadamente medio metro para realizar una nueva detección y se vuelve a analizar los resultados.

En la última posición del módulo de la Figura 4.5 se puede observar que este se encuentra a la mitad de su recorrido, luego de simular el movimiento del robot dentro de la habitación, y su respectivo archivo generado con la información del objeto detectado. En el tercer procesamiento se simula la evasión de obstáculos del sistema de navegación, ya que como se ve en la Figura 4.5 existe una camilla en la trayectoria entre el paciente y el robot, de esta manera, el robot rodea la camilla avanzando hasta llegar a la Figura 4.7. El hecho que los valores se repitan en la orientación es porque al evaluar nuevamente se encuentra en el mismo intervalo porque es discretizada. Luego del envío de parámetros se mueve el robot de acuerdo a la discretización se gira el módulo 17.5° en sentido antihorario y avanza aproximadamente medio metro para realizar una nueva detección y se vuelve a analizar los resultados de orientación y distancia.

Figura 4.7: Segunda secuencia de imágenes adquiridas por el sistema.

En la Figura 4.7 se muestra la segunda etapa de la secuencia de imágenes obtenida respecto al tiempo el cual simula el movimiento del robot hasta lograr posicionarse frente a la persona mediante el uso del módulo fabricado mostrado en la Figura 3.7 y la

información publicada de acuerdo a la categoría seleccionada, se logra posicionar al robot a 0.5 m del paciente ($\sqrt{x^2 + y^2} = \sqrt{(-0.0435)^2 + 0.49809^2} = 0.5\ m$).

Luego de simular el movimiento, con su respectivo archivo generado con la información del objeto detectado mostrado en la imagen se ejecuta las predicciones de distancia y orientación que ofrece como resultado la tercera pose mostrada en la Figura 4.5. A diferencia del procesamiento anterior, no existe obstáculo entre la simulación de la trayectoria es por ello que una vez rotado en sentido antihorario se muestra en la determinación de parámetros un ángulo de rotación de 0°, cuarto intervalo de la Tabla 3.1. Por lo tanto, luego del envío de parámetros el robot avanza aproximadamente medio metro para realizar una nueva detección y se vuelve a analizar los resultados de orientación y distancia. En estos cuatro primeros procesamientos el sistema es capaz de detectar tanto la camilla como a la persona dentro de la habitación, además para los fines de esta tesis se obvia los otros datos de camillas detectadas ya que como se menciona en el primer capítulo en la habitación solo habrá una camilla y un paciente echado sobre ella. Adicionalmente, tanto en la Figura 4.5 y 4.7 se presenta la secuencia de información relevante enviada al tópico de lectura realizado por la ejecución del sistema de localización. Finalmente, una vez ubicado en esta posición se procede a realizar otras funciones del robot, como presentación del robot, saludo del mismo, videoconferencia, entre otras. El algoritmo seguirá ejecutándose paralelamente a otras funciones del proyecto mientras se use la cámara, ya que solo requiere una imagen, toda esta información se enviará por la red local que cubre a la Jetson Nano y Xavier.

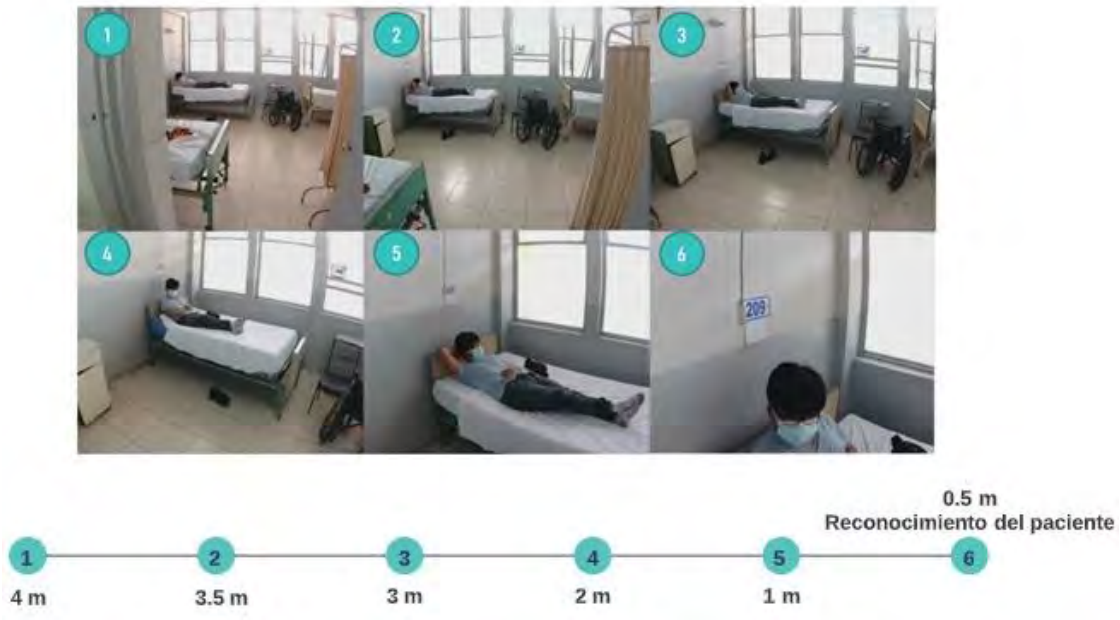


Figura 4.8: Resumen de la operación real del sistema.

4.5 ANÁLISIS DE LOS RESULTADOS

En esta sección se presenta la justificación matemática en forma de gráficos de los resultados en simulación realizados en una habitación de hospital. Considerando que el sistema se basa en dos modelos principales pre entrenados, se detalla el algoritmo de entrenamiento en el Anexo 3. De acuerdo a la metodología de validación explicada en la sección 3.5 se muestra el gráfico de la Figura 4.9 que representa la precisión del entrenamiento y validación del modelo de orientación; como se observa el sistema tiene aproximadamente 0.99 de precisión en la validación y 0.80 en el entrenamiento dentro del dataset creado con datos reales y aleatorios contando con un total de 3500 datos.

Asimismo, en la Figura 4.10 se muestra la representación de la pérdida de valores procesados tanto en la validación y entrenamiento los cuales representan que tan bien se ajustan para los datos nuevos y entrenados respectivamente.

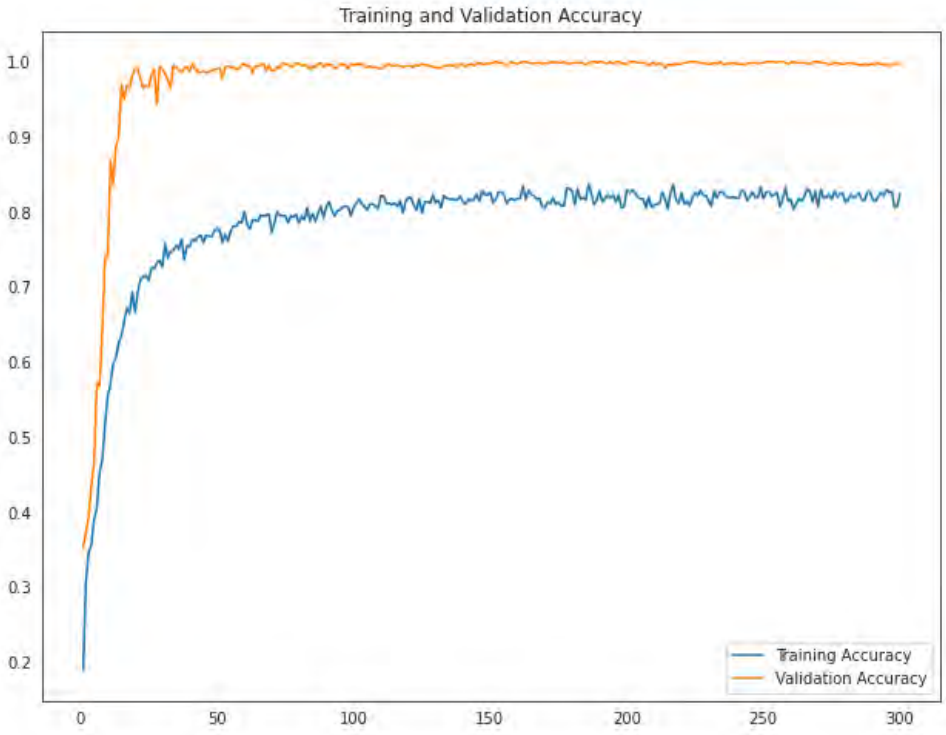


Figura 4.9: Precisión del entrenamiento y validación del modelo de orientación.

Ambas figuras son el resultado de 300 iteraciones de entrenamiento del algoritmo logrado en un período de 2 minutos, fuera del proceso del algoritmo, ya que es un pre entrenamiento, no afecta en la duración del desarrollo del sistema de localización.

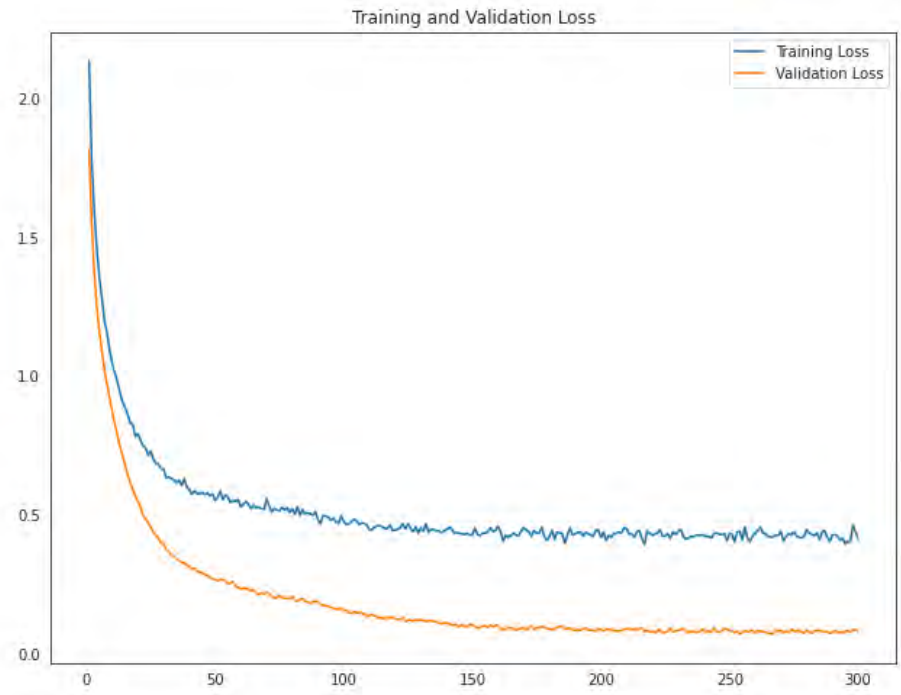


Figura 4.10: Pérdidas del entrenamiento y validación del modelo de orientación.

Finalmente se muestra la Figura 4.11 se muestra la gráfica de la precisión del entrenamiento y la validación para el modelo de predicción de distancia.

Figura 4.11: Precisión del entrenamiento y validación del modelo de distancia.

Como se observa en la figura se tiene 0.82 de valor en la precisión de la validación y un valor de 0.52 para el entrenamiento, estos valores sin el reflejo de distintas

modificaciones realizadas en la generación de este modelo dado que realizar la predicción de distancia por imágenes se debe tener en consideración lo siguiente:

- La mayoría de veces se detecta parte de la persona, lo que altera el área del cuadro delimitador y asignación del mismo.
- Los valores de precisión están sujetos a que se detecte tanto a la persona y la camilla en el mejor de los casos, el cual se da a una distancia mayor a dos metros, en distancias menores a ellas, la camilla deja de detectarse.
- Se puede mejorar obteniendo una mayor recolección del dataset específicamente para la predicción de distancia, ya que se usa los mismos 3500 valores usados en la orientación pues para diez intervalos no es suficiente y agregar más datos aleatorios disminuye la precisión del algoritmo.
- Los resultados obtenidos en la trayectoria real en la sección 4.4 son el resultados de las mejores muestras en el desarrollo del sistema dentro de la habitación.

Adicionalmente, para la obtención de los resultados reales se tomaron las siguientes consideraciones.

- En las Figuras 4.5 y 4.7 se muestran poses publicadas en ROS para el correcto posicionamiento según el trayecto realizado por el módulo hasta llegar a medio metro o menos de distancia entre el robot y el paciente, esta distancia es determinada también por los sensores de proximidad incorporados en el sistema de navegación.
- Los sensores determinaran que tanto debe acercarse además de poder evadir los obstáculos presentes en la trayectoria; en este caso, se simuló manualmente este sistema para que no coalicione con otro objeto.
- Finalmente, se obviaron objetos adicionales detectados, como es el caso de camillas adicionales; ya que, para los fines de la tesis se considera una habitación ideal de hospital con solo una camilla de reposo y un paciente sobre este, como se menciona en el primer capítulo.

CONCLUSIONES

- A partir de los requerimientos del proyecto de investigación, se diseñó el diagrama electrónico de adquisición y procesamiento de imágenes, asimismo se verificaron los componentes para la captura de imágenes, computador de alto rendimiento y comunicación con el computador principal. Al realizar el balance de consumo energético de los componentes, se demostró el funcionamiento adecuado del sistema.
- De acuerdo a los ensayos experimentales realizados en habitaciones UCI de un centro de salud, el algoritmo de detección de reconocimiento de pacientes y camillas basado en YOLO, no presentó ningún caso en que no se detecte al paciente con una precisión entre el 95 a 99.9%. Por el contrario, en la detección de la camilla, para una distancia superior a los dos metros decae su precisión de 99.9% a 50%.
- De acuerdo a los ensayos experimentales realizados en habitaciones UCI de un centro de salud, se comprobó que el algoritmo de localización del paciente con respecto al robot tiene una precisión de 99.12% respecto a la orientación y una precisión de 81.99% respecto a la distancia, estas precisiones son resultados de las predicciones del 20% de datos seleccionados dentro del dataset ".xlsx" considerando datos reales y aleatorios, tomando en cuenta una mayor parte de datos reales recolectados por el módulo de pruebas.
- Se demuestra que el detector de objetos basado en una arquitectura YOLO puede comunicarse con un sistema basado en entorno ROS, lo que garantiza una comunicación con el sistema de navegación del robot mediante tópicos y nodos de tipo maestros o esclavos determinados por el sistema. Lo anterior es determinante para la comunicación con la Jetson Xavier de tal manera que este se ejecute como maestro recibiendo información del nodo esclavo en la Jetson Nano.
- La emisión de la pose en un publicador por la Jetson Nano se estructura en forma de *"quaternion"* y coordenadas para la rotación y traslación del robot. La Xavier se subscribe a este tópico, adquiere la pose y ejecuta el movimiento respectivo.
- Mediante las simulaciones y pruebas experimentales realizadas mostradas en el último capítulo de la tesis se concluye que el algoritmo de localización logra enviar los parámetros de trayectorias para que el robot pueda posicionarse hasta que sea posible reconocer visualmente el rostro del paciente dentro de una habitación mediante su sistema de movimiento y evasión de obstáculos implementados en el sistema de navegación integrado en el robot.

RECOMENDACIONES

- El reconocimiento facial realizado mediante el algoritmo YOLO implicado por la detección del paciente que se genera en la ubicación a 0.5 metros del paciente, el cual se ve el rostro es su totalidad, para la asignación de este rostro a una persona en específica se espera crear un algoritmo en torno a la base de datos de paciente donde el robot realizará sus funciones en el futuro.
- Poder tener una base de datos con mayor cantidad de datos ayudará a obtener una mejor precisión del algoritmo en situaciones reales durante su ejecución en el hospital, especialmente se recomienda la toma de datos reales de distancia del paciente respecto al robot, ya que con los valores aleatorios generados si bien podemos realizar una trayectoria correcta a un 0.52 de precisión en la distancia y 0.82 en la orientación no es suficiente preciso si se somete a cambio de ambientes con diferentes características.
- El detector de objetos YOLO tiene la posibilidad de acoplarse en ROS2, desde la detección de imágenes por nodos hasta la generación del cuadro delimitador, por lo que para futuras aplicaciones se considerará el uso de esto para agilizar el período de ejecución de trayectoria dentro del computador Jetson Nano.
- Con la finalidad de lograr una ejecución predictiva rápida y más precisa de la trayectoria generada, es decir, para obtener una resolución de 20 o 30 cm dentro de una base de datos de mayor extensión. Se recomienda generar mayor números de intervalos.
- Se recomienda crear el propio detector específico para este tipo de pacientes insertando miles de cuadros delimitadores de miles de casos reales para que el algoritmo YOLO cree su clasificación interna y detecta lo necesario, ya que en este caso se hace una detección de personas y camillas por el contexto social.

REFERENCIA BIBLIOGRÁFICA

[1] C. A. Denckla, B. Gelaye, L. Orlinsky, and K. C. Koenen, “REACH for mental health in the COVID19 pandemic: an urgent call for public health action,” *European Journal of Psychotraumatology*, vol. 11, no. 1. Taylor and Francis Ltd., Dec. 31, 2020. doi: 10.1080/20008198.2020.1762995.

[2] J. Ruiz-Del-Solar *et al.*, “Mental and Emotional Health Care for COVID-19 Patients: Employing Pudu, a Telepresence Robot,” *IEEE Robotics and Automation Magazine*, vol. 28, no. 1, pp. 82–89, Mar. 2021, doi: 10.1109/MRA.2020.3044906.

[3] C. Liu, “The development trend of evaluating face-recognition technology,” in *Proceedings - 2014 International Conference on Mechatronics and Control, ICMC 2014*, Aug. 2015, pp. 1540–1544. doi: 10.1109/ICMC.2014.7231817.

[4] M. Wang and W. Deng, “Deep Face Recognition: A Survey,” Apr. 2018, doi: 10.1016/j.neucom.2020.10.081.

[5] H. Durrant-Whyte and T. Bailey, “Simultaneous localization and mapping: Part I,” *IEEE Robotics and Automation Magazine*, vol. 13, no. 2, pp. 99–108, Jun. 2006, doi: 10.1109/MRA.2006.1638022.

[6] T. Bailey, J. Nieto, J. Guivant, M. Stevens, and E. Nebot, “Consistency of the EKF-SLAM algorithm,” in *IEEE International Conference on Intelligent Robots and Systems*, 2006, pp. 3562–3568. doi: 10.1109/IROS.2006.281644.

[7] W. Liu, T. Wang, and Y. Zhang, “A relative map approach for efficient EKF-SLAM,” in *2014 IEEE Chinese Guidance, Navigation and Control Conference, CGNCC 2014*, Jan. 2015, pp. 2646–2650. doi: 10.1109/CGNCC.2014.7007586.

[8] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, “ORB-SLAM: A Versatile and Accurate Monocular SLAM System,” *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, Oct. 2015, doi: 10.1109/TRO.2015.2463671.

[9] “ARI - PAL Robotics: Leading service robotics.” https://pal-robotics.com/es/robots/ari/ (accessed May 21, 2021).

[10] T. Bailey, J. Nieto, and E. Nebot, “Consistency of the FastSLAM algorithm,” in *Proceedings - IEEE International Conference on Robotics and Automation*, 2006, vol. 2006, pp. 424–429. doi: 10.1109/ROBOT.2006.1641748.

[11] S. Nagla, “2D Hector SLAM of Indoor Mobile Robot using 2D Lidar,” Dec. 2020. doi: 10.1109/ICPECTS49113.2020.9336995.

[12] M. Filipenko and I. Afanasyev, “Comparison of Various SLAM Systems for Mobile Robot in an Indoor Environment,” in *9th International Conference on Intelligent Systems 2018: Theory, Research and Innovation in Applications, IS 2018 - Proceedings*, Jul. 2018, pp. 400–407. doi: 10.1109/IS.2018.8710464.

[13] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha, “Visual simultaneous localization and mapping: a survey,” *Artificial Intelligence Review*, vol. 43, no. 1, pp. 55–81, Jan. 2015, doi: 10.1007/s10462-012-9365-8.

[14] “XR-1 Service Robot | CloudMinds Cloud Robots.” https://www.en.cloudminds.com/home-new/cloud-robots/xr-1/ (accessed May 11, 2021).

[15] M. Montemerlo, W. Whittaker, S. Thrun, A. Stentz, and D. Fox, “FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem With Unknown Data Association,” University of Washington, Pittsburgh, 2003.

[16] M. Grupp, P. Kopp, P. Huber, and M. Rätsch, “A 3D Face Modelling Approach for Pose-Invariant Face Recognition in a Human-Robot Environment,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9776 LNAI, pp. 121–134, Jun. 2016, Accessed: May 03, 2021. [Online]. Available: http://arxiv.org/abs/1606.00474

[17] “Solución Robótica para Negocios Tokyo the Robot.” Accessed: May 21, 2021. [Online]. Available: https://grupoadd.es/wp-content/uploads/2020/06/Tokyo-the-Robot-Soluciones-para-Negocios-FuturaVIVE-GrupoADD-V26JUNIO2020-4.pdf

[18] “ARI TECHNICAL SPECIFICATIONS.” https://pal-robotics.com/wp-content/uploads/2020/06/ari-datasheet.pdf (accessed May 21, 2021).

[19] e-con Systems, “Getting Started with Sony Starvis IMX415 Camera for Jetson Xavier NX Developer Kit | e-con Systems - YouTube,” Feb. 03, 2021. https://www.youtube.com/watch?v=2j_uk83vANg (accessed May 03, 2021).

[20] e-con Systems, “Getting started 13MP (4K) Ultra HD Camera for NVIDIA Jetson Xavier NX / Jetson Nano | e-con Systems - YouTube,” Jan. 12, 2021. https://www.youtube.com/embed/bJTG242y3B0?controls=1 (accessed May 03, 2021).

[21] “(1) Getting started with e-CAM82_USB - SONY STARVIS™ IMX415 low light 4K USB 2.0 Camera | e-con Systems - YouTube.” https://www.youtube.com/watch?v=SxmF_CtETXQ (accessed May 03, 2021).

[22] “Course | Robotics: Vision Intelligence and Machine Learning | edX.” https://learning.edx.org/course/course-v1:PennX+ROBO2x+2T2017/home (accessed Jul. 04, 2021).

[23] C. C. Aggarwal, “Convolutional Neural Networks,” in *Neural Networks and Deep Learning*, Cham: Springer International Publishing, 2018, pp. 315–371. doi: 10.1007/978-3-319-94463-0_8.

[24] “CS231n Convolutional Neural Networks for Visual Recognition.” https://cs231n.github.io/convolutional-networks/ (accessed Jul. 04, 2021).

[25] “Neural Networks (Computational Intelligence) - Mr. Stevenson´s ITGS Classroom.” https://sites.google.com/site/mrstevensonstechclassroom/hl-topics-only/4a-robotics-ai/neural-networks-computational-intelligence (accessed Jul. 04, 2021).

[26] “A Neural Network Playground.” https://playground.tensorflow.org/ (accessed Jul. 04, 2021).

[27] A. Hidaka and T. Kurita, “Consecutive Dimensionality Reduction by Canonical Correlation Analysis for Visualization of Convolutional Neural Networks,” *Proceedings of the ISCIE International Symposium on Stochastic Systems Theory and its Applications*, vol. 2017, no. 0, pp. 160–167, May 2017, doi: 10.5687/sss.2017.160.

[28] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation Tech report (v5).” Accessed: Jul. 04, 2021. [Online]. Available: http://www.cs.berkeley.edu/˜rbg/rcnn.

[29] R. Girshick, “Fast R-CNN.” Accessed: Jul. 04, 2021. [Online]. Available: https://github.com/rbgirshick/

[30] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.” Accessed: Jul. 04, 2021. [Online]. Available: http://image-net.org/challenges/LSVRC/2015/results

[31] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection.” Accessed: Jul. 04, 2021. [Online]. Available: http://pjreddie.com/yolo/

[32] A. Boschi, F. Salvetti, V. Mazzia, and M. Chiaberge, “A cost-effective person-following system for assistive unmanned vehicles with deep learning at the edge,” *Machines*, vol. 8, no. 3, Sep. 2020, doi: 10.3390/MACHINES8030049.

[33] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection.” Accessed: Jun. 18, 2021. [Online]. Available: https://github.com/AlexeyAB/darknet.

ANEXO 1

Se muestra las modificaciones dentro del código implementado en lenguaje Python para lograr conectar la cámara e-CAM82_USB al computador Jetson Nano a 30 fps.

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

import cv2

cap = cv2.VideoCapture("v4l2src device=/dev(video0 ! video/x-raw,format=YUY2,width=640,height=480,framerate=30/1 ! nvvideconv ! video/x-raw(memory_NVMM) ! nvvidconv ! video/x-raw, format=BGRx ! videoconvert ! video/x-raw, format=BGR ! appsink drop=1", cv2.CAP_GSTREAMER)
```

ANEXO 2

Se muestra el código implementado en lenguaje Python para lograr cargar los dos modelos pre entrenados y el dataset con todos los datos recolectados dentro de la tabla en Excel, para asignar a las variables dentro del algoritmo y poder predecir la orientación y la distancia respecto al archivo ".json" cargado por el clasificador YOLO.

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

import tensorflow as tf

import pickle

import time

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error

from sklearn.base import clone

from sklearn.tree import DecisionTreeClassifier

import math

from tensorflow import keras

from tensorflow.keras import layers

import rospy

import rosbag

import numpy as np

import pandas as pd
```

```
import json
from std_msgs.msg import String
from geometry_msgs.msg import Quaternion

def asig_out(predictions,tamaño):
    print(predictions.round(2))
    print(tamaño.round(2))
    if predictions[0][1] > 0.7:
        giro = np.deg2rad(-47.5)
        print("debe girar papu : " , giro)
        if tamaño[0][1] > 0.4:
            print("Posicion perfecta del robot")
            referencia = 0.5
        else :
            if tamaño[0][2] > 0.4:
                print("Esta a un metro de distancia")
                referencia = 1
            else :
                if tamaño[0][3] > 0.4:
                    print("Esta a un metro y medio de distancia")
                    referencia = 1.5
                else:
                    if tamaño[0][4] > 0.4:
                        print("Esta a dos metros de distancia")
                        referencia = 2
                    else:
                        if tamaño[0][5] > 0.4:
                            print("Esta a dos metros y medio de distancia")
                            referencia = 2.5
                        else:
                            if tamaño[0][6] > 0.4:
```

```
                print("Esta a tres metros de distancia")
                referencia = 3
            else:
                if tamaño[0][7] > 0.4:
                    print("Esta a tres metros y medio de distancia")
                    referencia = 3.5
                else:
                    if tamaño[0][8] > 0.4:
                        print("Esta a cuatro metros de distancia")
                        referencia = 4
                    else:
                        if tamaño[0][9] > 0.4:
                            print("Esta a cuantro metros y medio de distancia")
                            referencia = 4.5
                        else:
                            referencia = 5
if predictions[0][2] > 0.7:
    giro = np.deg2rad(-32.5)
    print("debe girar papu : " , giro)
    if tamaño[0][1] > 0.4:
        print("Posicion perfecta del robot")
        referencia = 0.5
    else :
        if tamaño[0][2] > 0.4:
            print("Esta a un metro de distancia")
            referencia = 1
        else :
            if tamaño[0][3] > 0.4:
                print("Esta a un metro y medio de distancia")
                referencia = 1.5
            else:
```

```
            if tamaño[0][4] > 0.4:
                print("Esta a dos metros de distancia")
                referencia = 2
            else:
                if tamaño[0][5] > 0.4:
                    print("Esta a dos metros y medio de distancia")
                    referencia = 2.5
                else:
                    if tamaño[0][6] > 0.4:
                        print("Esta a tres metros de distancia")
                        referencia = 3
                    else:
                        if tamaño[0][7] > 0.4:
                            print("Esta a tres metros y medio de distancia")
                            referencia = 3.5
                        else:
                            if tamaño[0][8] > 0.4:
                                print("Esta a cuatro metros de distancia")
                                referencia = 4
                            else:
                                if tamaño[0][9] > 0.4:
                                    print("Esta a cuatro metros y medio de distancia")
                                    referencia = 4.5
                                else:
                                    referencia = 5
if predictions[0][3] > 0.7:
    giro = np.deg2rad(-17.5)
    print("debe girar papu : " , giro)
    if tamaño[0][1] > 0.4:
        print("Posicion perfecta del robot")
        referencia = 0.5
```

```
else :
    if tamaño[0][2] > 0.4:
        print("Esta a un metro de distancia")
        referencia = 1
    else :
        if tamaño[0][3] > 0.4:
            print("Esta a un metro y medio de distancia")
            referencia = 1.5
        else:
            if tamaño[0][4] > 0.4:
                print("Esta a dos metros de distancia")
                referencia = 2
            else:
                if tamaño[0][5] > 0.4:
                    print("Esta a dos metros y medio de distancia")
                    referencia = 2.5
                else:
                    if tamaño[0][6] > 0.4:
                        print("Esta a tres metros de distancia")
                        referencia = 3
                    else:
                        if tamaño[0][7] > 0.4:
                            print("Esta a tres metros y medio de distancia")
                            referencia = 3.5
                        else:
                            if tamaño[0][8] > 0.4:
                                print("Esta a cuatro metros de distancia")
                                referencia = 4
                            else:
                                if tamaño[0][9] > 0.4:
                                    print("Esta a cuatro metros y medio de distancia")
```

```
                    referencia = 4.5
                else:
                    referencia = 5
if predictions[0][4] > 0.7:
    giro = 0.0
    print("no debe girar nd " , giro)
    if tamaño[0][1] > 0.4:
        print("Posicion perfecta del robot")
        referencia = 0.5
    else :
        if tamaño[0][2] > 0.4:
            print("Esta a un metro de distancia")
            referencia = 1
        else :
            if tamaño[0][3] > 0.4:
                print("Esta a un metro y medio de distancia")
                referencia = 1.5
            else:
                if tamaño[0][4] > 0.4:
                    print("Esta a dos metros de distancia")
                    referencia = 2
                else:
                    if tamaño[0][5] > 0.4:
                        print("Esta a dos metros y medio de distancia")
                        referencia = 2.5
                    else:
                        if tamaño[0][6] > 0.4:
                            print("Esta a tres metros de distancia")
                            referencia = 3
                        else:
                            if tamaño[0][7] > 0.4:
```

```
                    print("Esta a tres metros y medio de distancia")
                    referencia = 3.5
                else:
                    if tamaño[0][8] > 0.4:
                        print("Esta a cuatro metros de distancia")
                        referencia = 4
                    else:
                        if tamaño[0][9] > 0.4:
                            print("Esta a cuatro metros y medio de distancia")
                            referencia = 4.5
                        else:
                            referencia = 5
if predictions[0][5] > 0.7:
    giro = np.deg2rad(17.5)
    print("debe girar papu : " , giro)
    if tamaño[0][1] > 0.4:
        print("Posicion perfecta del robot")
        referencia = 0.5
    else :
        if tamaño[0][2] > 0.4:
            print("Esta a un metro de distancia")
            referencia = 1
        else :
            if tamaño[0][3] > 0.4:
                print("Esta a un metro y medio de distancia")
                referencia = 1.5
            else:
                if tamaño[0][4] > 0.4:
                    print("Esta a dos metros de distancia")
                    referencia = 2
                else:
```

```
            if tamaño[0][5] > 0.4:
                print("Esta a dos metros y medio de distancia")
                referencia = 2.5
            else:
                if tamaño[0][6] > 0.4:
                    print("Esta a tres metros de distancia")
                    referencia = 3
                else:
                    if tamaño[0][7] > 0.4:
                        print("Esta a tres metros y medio de distancia")
                        referencia = 3.5
                    else:
                        if tamaño[0][8] > 0.4:
                            print("Esta a cuatro metros de distancia")
                            referencia = 4
                        else:
                            if tamaño[0][9] > 0.4:
                                print("Esta a cuatro metros y medio de distancia")
                                referencia = 4.5
                            else:
                                referencia = 5
if predictions[0][6] > 0.7:
    giro = np.deg2rad(32.5)
    print("debe girar papu : " , giro)
    if tamaño[0][1] > 0.4:
        print("Posicion perfecta del robot")
        referencia = 0.5
    else :
        if tamaño[0][2] > 0.4:
            print("Esta a un metro de distancia")
            referencia = 1
```

```
        else :
            if tamaño[0][3] > 0.4:
                print("Esta a un metro y medio de distancia")
                referencia = 1.5
            else:
                if tamaño[0][4] > 0.4:
                    print("Esta a dos metros de distancia")
                    referencia = 2
                else:
                    if tamaño[0][5] > 0.4:
                        print("Esta a dos metros y medio de distancia")
                        referencia = 2.5
                    else:
                        if tamaño[0][6] > 0.4:
                            print("Esta a tres metros de distancia")
                            referencia = 3
                        else:
                            if tamaño[0][7] > 0.4:
                                print("Esta a tres metros y medio de distancia")
                                referencia = 3.5
                            else:
                                if tamaño[0][8] > 0.4:
                                    print("Esta a cuatro metros de distancia")
                                    referencia = 4
                                else:
                                    if tamaño[0][9] > 0.4:
                                        print("Esta a cuatro metros y medio de distancia")
                                        referencia = 4.5
                                    else:
                                        referencia = 5
if predictions[0][7] > 0.7:
```

```
giro = np.deg2rad(47.5)
print("debe girar papu : " , giro)
if tamaño[0][1] > 0.4:
   print("Posicion perfecta del robot")
   referencia = 0.5
else :
   if tamaño[0][2] > 0.4:
      print("Esta a un metro de distancia")
      referencia = 1
   else :
      if tamaño[0][3] > 0.4:
         print("Esta a un metro y medio de distancia")
         referencia = 1.5
      else:
         if tamaño[0][4] > 0.4:
            print("Esta a dos metros de distancia")
            referencia = 2
         else:
            if tamaño[0][5] > 0.4:
               print("Esta a dos metros y medio de distancia")
               referencia = 2.5
            else:
               if tamaño[0][6] > 0.4:
                  print("Esta a tres metros de distancia")
                  referencia = 3
               else:
                  if tamaño[0][7] > 0.4:
                     print("Esta a tres metros y medio de distancia")
                     referencia = 3.5
                  else:
                     if tamaño[0][8] > 0.4:
```

```
                print("Esta a cuatro metros de distancia")
                referencia = 4
            else:
                if tamaño[0][9] > 0.4:
                    print("Esta a cuantro metros y medio de distancia")
                    referencia = 4.5
                else:
                    referencia = 5
    return giro, referencia


def asig_modelo():
    data=pd.read_excel("Libro5.xlsx")
    x = data.drop(["target","target2"], axis=1)
    y = data["target"]
    y2 = data["target2"]
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42,
stratify=y)
    x2_train, x2_test, y2_train, y2_test = train_test_split(x, y2, test_size=0.2,
random_state=42, stratify=y)
    def dataframe_to_dataset(df_x, df_y):
        ds = tf.data.Dataset.from_tensor_slices((dict(df_x), df_y))
        ds = ds.shuffle(buffer_size=len(df_x))
        return ds
    #Se generan los conjuntos de datos de entrenamiento y validación a formato de
tensor flow
    train_ds = dataframe_to_dataset(x_train, y_train)
    val_ds = dataframe_to_dataset(x_test, y_test)
    train_ds = train_ds.batch(14)
    val_ds = val_ds.batch(14)
    model = keras.models.load_model('modelo_entrenado_orientacion.h5')
    model.fit(train_ds, epochs=3, validation_data=val_ds)
    train_ds = dataframe_to_dataset(x2_train, y2_train)
```

```
    val_ds = dataframe_to_dataset(x2_test, y2_test)
    train_ds = train_ds.batch(14)
    val_ds = val_ds.batch(14)
    model2 = keras.models.load_model('modelo_entrenado_distancia.h5')
    model2.fit(train_ds, epochs=3, validation_data=val_ds)
    return model, model2

def red_neuronal(model,model2):
    with open('pose.json') as file:
        data = json.load(file)
    #EXTRACCIÓN DE DATOS: PERSON
    for i in range (0,len(data[0]['objects'])):
        if data[0]['objects'][i]['class_id'] == 0 :
            centro = data[0]['objects'][i]['relative_coordinates']['center_x']
            ancho = data[0]['objects'][i]['relative_coordinates']['width']
            largo = data[0]['objects'][i]['relative_coordinates']['height']
            tamaño = ancho*largo*100*100
            centro2=centro
            ancho2=ancho
            largo2=largo
            tamaño2=tamaño
            #print('center_X:', data[0]['objects'][i]['relative_coordinates']['center_x'])
    #EXTRACCIÓN DE DAROS BED
    for i in range (0,len(data[0]['objects'])):
        if data[0]['objects'][i]['class_id'] == 60 :
            centro2 = data[0]['objects'][i]['relative_coordinates']['center_x']
            ancho2 = data[0]['objects'][i]['relative_coordinates']['width']
            largo2 = data[0]['objects'][i]['relative_coordinates']['height']
            tamaño2 = ancho2*largo2*100*100
            #print('center_X:', data[0]['objects'][i]['relative_coordinates']['center_x'])
    sample = {
```

```
    "tamaño": 1,
    "x1": 100*centro,
    "x2": 100*centro2,
    "y1": 50,
    "y2": 50,
    "centrox": (100*centro+100*centro2)*0.5,
    "centroy": 55,
    "difx": tamaño,
    "dify": tamaño2,
  }
  input_dict = {name: tf.convert_to_tensor([value]) for name, value in sample.items()}
  predictions = model.predict(input_dict)
  predictions2 = model2.predict(input_dict)
  return predictions, predictions2

def simplePublisher(distancia,angulo):
  simple_publisher = rospy.Publisher('topic_1', Quaternion, queue_size = 10)
  rospy.init_node('node_1', anonymous = False)
  rate = rospy.Rate(10)
  topic1_content = Quaternion(
      w=angulo,
      x=distancia*math.sin(angulo),
      y=distancia*math.cos(angulo),
      z=0.7
  )
  simple_publisher.publish(topic1_content)
  #rate.sleep()

if __name__ == '__main__':
  model, model2 = asig_modelo()
  try:
```

```
    count=5
    while (count>0) :
        predictions, tamaño = red_neuronal(model, model2)
        giro, referencia = asig_out(predictions, tamaño)
        simplePublisher(referencia,giro)
        time.sleep(5)
        count=count-1
  except rospy.ROSInterruptException:
    pass
```

ANEXO 3

Se muestra el código del pre entrenamiento para ambos modelos en lenguaje Python usando los datos recolectados por la cámara e-CAM82_USB al computador Jetson Nano en la tabla Excel generada de ellos además de datos aleatorios relacionados. Finalmente se obtiene ambos modelos (".h5") los cuales se cargan en el Anexo 2.

```
import pickle
import pandas as pd
import numpy as np
import os
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.base import clone
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
sns.set_style(style='white')
data=pd.read_excel("Libro5.xlsx")
data.head()
x = data.drop(["target","target2"], axis=1)
```

```
#x = data.drop(["target"], axis=1)

y = data["target2"]

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42, stratify=y)

#Se genera la función "tf.data.Dataset" para convertir dataframe de panda a dataset de tensor flow

def dataframe_to_dataset(df_x, df_y):

    ds = tf.data.Dataset.from_tensor_slices((dict(df_x), df_y))

    ds = ds.shuffle(buffer_size=len(df_x))

    return ds

#Se generan los conjuntos de datos de entrenamiento y validación a formato de tensor flow

train_ds = dataframe_to_dataset(x_train, y_train)

val_ds = dataframe_to_dataset(x_test, y_test)

from tensorflow.keras.layers.experimental.preprocessing import IntegerLookup

from tensorflow.keras.layers.experimental.preprocessing import Normalization

from tensorflow.keras.layers.experimental.preprocessing import StringLookup

def encode_numerical_feature(feature, name, dataset):

    # Crear la capa de Normalization para las características

    normalizer = Normalization()

    # Preparar el conjunto de datos con solo nuestros datos de interés

    feature_ds = dataset.map(lambda x, y: x[name])

    feature_ds = feature_ds.map(lambda x: tf.expand_dims(x, -1))

    # Aprender las estadísticas de los datos

    normalizer.adapt(feature_ds)

    # Normalizalizar la función de entrada

    encoded_feature = normalizer(feature)

    return encoded_feature

def encode_categorical_feature(feature, name, dataset, is_string):

    lookup_class = StringLookup if is_string else IntegerLookup

    # Crear una capa de búsqueda que convierta caracteres en enteros

    lookup = lookup_class(output_mode="binary")
```

```
    # Preparar el conjunto de datos con solo nuestros datos de interés
    feature_ds = dataset.map(lambda x, y: x[name])
    feature_ds = feature_ds.map(lambda x: tf.expand_dims(x, -1))
    # Aprender los posibles valores de los caracteres y asignarles valores enteros
fijos
    lookup.adapt(feature_ds)
    # Cambiar los caracteres de entrada a indices enteros
    encoded_feature = lookup(feature)
    return encoded_feature
tamaño = keras.Input(shape=(1,), name="tamaño")
x1 = keras.Input(shape=(1,), name="x1")
x2 = keras.Input(shape=(1,), name="x2")
y1 = keras.Input(shape=(1,), name="y1")
y2 = keras.Input(shape=(1,), name="y2")
centrox = keras.Input(shape=(1,), name="centrox")
centroy = keras.Input(shape=(1,), name="centroy")
difx = keras.Input(shape=(1,), name="difx")
dify = keras.Input(shape=(1,), name="dify")
all_inputs = [
    tamaño,
     x1,
     x2,
     y1,
     y2,
     centrox,
     centroy,
     difx,
     dify,
]
tamaño_encoded = encode_numerical_feature(tamaño, "tamaño", train_ds)
x1_encoded = encode_numerical_feature(x1, "x1", train_ds)
```

```
x2_encoded = encode_numerical_feature(x2, "x2", train_ds)
y1_encoded = encode_numerical_feature(y1, "y1", train_ds)
y2_encoded = encode_numerical_feature(y2, "y2", train_ds)
centrox_encoded = encode_numerical_feature(centrox, "centrox", train_ds)
centroy_encoded = encode_numerical_feature(centroy, "centroy", train_ds)
difx_encoded = encode_numerical_feature(difx, "difx", train_ds)
dify_encoded = encode_numerical_feature(dify, "dify", train_ds)
all_features = layers.concatenate(
   [
     tamaño_encoded,
     x1_encoded,
     x2_encoded,
     y1_encoded,
     y2_encoded,
     centrox_encoded,
     centroy_encoded,
     difx_encoded,
     dify_encoded,
 ]
)
# Configuración del modelo de red neuronal con una salida con activación "softmax"
cambiar para orientación y distancia respecivamente
x = layers.Dense(14, activation="relu")(all_features)
x = layers.Dropout(0.5)(x)
output = layers.Dense(11, activation="softmax")(x)
model = keras.Model(all_inputs, output)
model.compile("adam", "sparse_categorical_crossentropy", metrics=["accuracy"])
keras.utils.plot_model(model, show_shapes=True, rankdir="LR")
history = model.fit(train_ds, epochs=300, validation_data=val_ds)
```