EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de Chihuahua II

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

APLICACIÓN DE REDES NEURONALES Y VISIÓN ARTIFICIAL PARA LA NAVEGACIÓN AUTÓNOMA DE UN ROBOT MÓVIL

TESIS

PARA OBTENER EL GRADO DE

MAESTRO EN SISTEMAS COMPUTACIONALES

PRESENTA

LUIS RODRIGO BAÑUELOS HERNÁNDEZ

DIRECTOR DE TESIS

DR. RAFAEL SANDOVAL RODRÍGUEZ

CODIRECTOR DE TESIS

M.C. ARTURO LEGARDA SÁENZ

CHIHUAHUA, CHIH., MAYO 2023

Dictamen

Chihuahua, Chihuahua,02 de junio 2023

M.C. MARIA ELENA MARTINEZ CASTELLANOS

COORDINADORA DE POSGRADO E INVESTIGACION.

PRESENTE

Por medio de este conducto el comité tutorial revisor de la tesis para obtención de grado de Maestro en Sistemas Computacionales, que lleva el nombre de: "APLICACIÓN DE REDES NEURONALES Y VISIÓN ARTIFICIAL PARA LA NAVEGACIÓN AUTÓNOMA DE UN ROBOT MÓVIL", que presenta el C. LUIS RODRIGO BAÑUELOS HERNANDEZ, hace de su conocimiento que después de ser revisado ha dictaminado la APROBACIÓN de la misma.

Sin otro particular de momento queda de usted.

Atentamente

La Comisión de Revisión de Tesis.

DR. RAFAEL SANDOVAL RODRÍGUEZ

Director de tesis

M.C. ARTURO LEGARDA SÁENZ

Co –Director

DR. HERNÁN DE LA GARZA GUTIERREZ

Revisor

DR. ARTURO MARTÍNEZ AYALA

Revisor

DEDICATORIA

Este trabajo no habría sido posible sin el apoyo incondicional de mis padres, María y Martín, que no solo me alentaron a seguir este camino, sino que durante todo el proceso estuvieron para darme la fuerza, la paciencia, y la disciplina para cumplir con todos los objetivos en mi vida.

Así mismo, dedico este trabajo a todos los estudiantes, maestros e integrantes de las comunidades académicas, quienes son los que aprovecharán los conocimientos obtenidos en todos estos trabajos de investigación, ya sea como punto de partida para nuevos conocimientos como para descartar caminos sin salida.

Dedico esta tesis, a mis compañeros de generación, que estuvieron a mi lado durante todo el camino e hicieron este tiempo de lectura, trabajo, pruebas y errores, mucho más llevadero. Y por último, a mis maestros, quienes fueron lo suficientemente pacientes, dedicados y preparados como para ayudarme a completar este trabajo.

AGRADECIMIENTOS

No hay suficiente espacio en este documento para agradecer a todas y cada una de las personas que me ayudaron durante este trabajo, pero intentaré meter a todas las que pueda en los siguientes párrafos.

Primeramente, agradezco a mi familia, a mis padres María y Martín quienes me apoyaron desde el día uno a entrar al programa, a mis hermanos Teresa y Martín por levantarme el ánimo cada que se acercaba el final de semestre y los avances parecían inalcanzables.

Agradezco también a mi Alma Mater, el Instituto Tecnológico Nacional de México Campus Chihuahua II, por prestar sus instalaciones para realizar las pruebas finales de este proyecto, a sus increíbles docentes, entre los que puedo nombrar a mis asesores: el Dr. Rafael Sandoval y el Maestro Arturo Legarda, al Dr. Carlos Rubio Rascón por su guía imperdible para comprender como trabajar con redes neuronales y a toda la plantilla de posgrado por brindar apoyo de todas las maneras posibles para llevar este trabajo a su conclusión. Así mismo, agradezco a Conacyt por permitirme realizar este programa de maestría gracias a su programa de becas.

Por último, me gustaría agradecer a todos los amigos a los que conté, narré e incluso importuné hablando sobre mi proyecto de tesis, entre los cuales se encuentran Silvestre Lugo, Claudia Jurado, Daniel Nevarez e Ilse Olivas. Sin sus bromas, reuniones para despejar mi mente del trabajo y apoyo general este trabajo no habría sido posible.

Como mencioné en el primer párrafo de estos agradecimientos, la lista de personas a las que me gustaría agradecer es muy extensa, mi vida ha dado un giro impresionante desde que empecé este programa y les debo a todos un reconocimiento, a mi demás familia, amigos cercanos y no tan cercanos, a mis compañeros de carrera y de maestría. Al final del día, una de las cosas que aprendí con este trabajo y este proceso es que la vida no se trata de ser el mejor, sino de ser mejor que tu yo del día anterior.

RESUMEN

En esta tesis se presenta una propuesta para integrar un sistema para detectar las líneas que marcan los límites de circulación en el piso de producción, utilizando técnicas de inteligencia artificial como lo son la visión por computadora y las redes neuronales para lograr la identificación de los límites de manera efectiva y repetible. Para desarrollar este trabajo se utilizaron tecnologías tales como el lenguaje de programación Python con las librerías de OpenCV para trabajar la parte de visión por computadora y Tensorflow la red neuronal que involucra el algoritmo YOLOv5. Así mismo, se utilizó una cámara montada en un modelo de robot educativo para obtener las imágenes usadas en el entrenamiento del sistema, capturadas en un ambiente de pruebas controlado a escala, con el propósito de variar las configuraciones de los pasillos.

Los resultados de dichos algoritmos muestran no solo la efectividad del método propuesto, sino las comparaciones obtenidas entre variaciones de entrenamientos, y las diferencias de detección de varios métodos de detección de bordes.

ABSTRACT

This thesis presents a proposal to integrate a system to detect the lines that mark the limits of circulation on the production floor, using artificial intelligence techniques such as computer vision and neural networks to achieve the identification of the limits in an effective and repeatable way. To develop this work, technologies such as Python programming language with OpenCV libraries were used to work the computer vision part and Tensorflow the neural network that involves the YOLOv5 algorithm. Also, a camera mounted on a model of educational robot was used to obtain the images required for training the system, which were captured in a controlled test environment at scale, in order to vary the configurations of the aisles.

The results of these algorithms show not only the effectiveness of the proposed method, but the comparisons obtained between training variations, and the detection differences of various edge detection methods.

CONTENIDO

DICTAMEN ii
DEDICATORIA iii
AGRADECIMIENTOS iv
RESUMEN v
ABSTRACT v
INDICE DE FIGURAS viii
INDICE DE GRÁFICAS xi
CAPÍTULO I. INTRODUCCIÓN. 1
1.1 Introducción. 1
1.2 Planteamiento del problema. 2
1.3 Alcances y limitaciones. 3
1.4 Justificación. 3
1.5 Objetivos. 4
1.5.1 Objetivo general. 4
1.5.2 Objetivos específicos. 4
CAPÍTULO II. ESTADO DEL ARTE. 6
2.1 *Deep Hough transform for semantic line detection.* 6
2.2 *Iterative Hough Transform for line detection in 3D point clouds*. 7
2.3 *New approach for beacons based mobile robots localization using Kalman filters*. 7
2.4 *Power line detection using Hough transform and line tracing techniques*. 8
2.5 *Autonomous navigation of mobile robots in factory enviroments*. 9
2.6 *Comparison of three key remote sensing technologies for mobile robots locations in nuclear facilities*. 9
CAPÍTULO III. MARCO TEÓRICO. 14
3.1 Visión artificial. 14
3.1.1 Escala de grises. 14
3.1.2 Difuminado. 15
3.1.3 Binarización. 16
3.1.4 Operaciones morfológicas. 18
3.1.5 Función *Canny* 21

3.1.6 Transformada de Hough. 22
3.2 Redes neuronales. 25
3.2.1 Descripción general. 25
3.2.2 Entrenamiento de una red neuronal. 28
3.2.3 Funcionamiento de redes neuronales convolucionales. 30
3.3 YOLO: You Only Look Once. 32
CAPÍTULO IV. DESARROLLO. 40
4.1 Condiciones iniciales. 40
4.2 Imágenes de prueba. 41
4.3 Preprocesado de imágenes. 43
4.4 Procesado de imágenes. 44
4.4.1 Comparación *findContours* y *Houghlines*. 44
4.4.2 Comparación *Houghlines* con promediado y *HoughlinesP*. 45
4.4.3 Promediado de líneas. 46
4.4.4 Abstracción de líneas. 47
4.5 Entorno simulado para obtención de imágenes de entrenamiento. 48
4.6 Resultados del procesado 50
4.7 Etiquetado para entrenamiento de red neuronal. 52
4.8 Entrenamiento de red neuronal YOLOv5. 55
CAPÍTULO V. RESULTADOS Y DISCUCIÓN. 61
5.1 Resultados de entrenamiento. 61
5.1.1 Resultados con el primer *dataset* 61
5.1.2 Resultados con el segundo *dataset*. 64
5.1.3 Descripción de errores significativos. 65
CAPÍTULO VI. CONCLUSIONES. 68
CAPÍTULO VII. BIBLIOGRAFÍA. 70
ANEXOS. 72
Anexo A: Código fuente del sistema 72
Anexo B: Enlaces de herramientas utilizadas. 78
Anexo C: Tabla de resultados de la abstracción. 79
Anexo D: Tabla de resultado de entrenamiento. 81

INDICE DE FIGURAS

Figura 1.1 Vehículos autónomos guiados utilizados por Amazon. 2

Figura 1.2 Diagrama de sistemas ligados a un vehículo autónomo. 3

Figura 2.1 Distribución de los sensores para prueba de funcionamiento. 10

Figura 2.2 Mapa generado a partir de la información de los sensores. 11

Figura 3.1 a) Imagen original y b) Imágen convertida a escala de grises. 15

Figura 3.2 Resultado de la operación de Difuminado. 16

Figura 3.3 a) Binarización estándar, b) Binarización inversa y c) Binarización Otsu. 18

Figura 3.4 Resultado de la operación de Dilatación. 19

Figura 3.5 Resultado de la operación de Erosión. 20

Figura 3.6 a) Imagen base, b) Apertura y c) Clausura. 21

Figura 3.7 a) Imagen base y b) Resultado de Canny() 22

Figura 3.8 Líneas representadas por la transformada de Hough. 24

Figura 3.9 Caso de ambigüedad detectado en la abstracción. 24

Figura 3.10 Diagrama de una neurona biológica. 25

Figura 3.11 Diagrama de una neurona artificial. 26

Figura 3.12 Formas de funciones de activación. a) Lineal, b) Escalonada, c) Sigmoide y d) ReLU. 28

Figura 3.13 Diagrama de proceso de convolución. 31

Figura 3.14 Estructura de la red neuronal implementada en YOLO. 33

Figura 4.1 Diagrama general del sistema de visión. 40

Figura 4.2 Primera versión de imagen de prueba. 41

Figura 4.3 Segunda versión de imagen de prueba. 42

Figura 4.4 Entorno de captura de imágenes de prueba. 42

Figura 4.5 Resultados del preprocesado de imagen. a) Imagen base, b) Escala de grises, c) Difuminado, d) Binarización inversa + Otsu y e) Apertura + Clausura. 43

Figura 4.6 Resultados de operaciones: a) Canny y b) findContours. 44

Figura 4.7 líneas detectadas por HoughLines. 45

Figura 4.8 Comportamiento de HoughlinesP. 46

Figura 4.9 Representación de líneas promediadas. 47

Figura 4.10 Abstracción de las líneas reales obtenida por el algoritmo. 47

Figura 4.11 Caso de ambigüedad detectado en el proceso de abstracción. 48

Figura 4.12 Robot educacional para pruebas. a) Vista frontal, b) Vista lateral y c) Control RF. 49

Figura 4.13 Configuración de ambiente de pruebas. 49

Figura 4.14 Ambiente de pruebas configurado para captura de imágenes. 50

Figura 4.15 Diagrama del sistema de visión con un sistema de clasificación por red neuronal. 52

Figura 4.16 Captura de la página de Roboflow correspondiente a la primera versión del dataset. 53

Figura 4.17 Captura de la página de Roboflow correspondiente a la segunda versión del dataset. 54

Figura 4.18 Método de etiquetado para la primera versión del dataset. 54

Figura 4.19 Método de etiquetado para la segunda versión del dataset. 55

Figura 5.1 Gráficas de precisión máxima alcanzada con los diferentes tamaños de batch: a) 8, b) 16, c) 32 y d) 64. 62

Figura 5.2 a) Línea de código para parametrizar el entrenamiento de la red y b) Error de memoria de la librería Keras. 64

Figura 5.3 Inconsistencia en la abstracción: a) Imagen real y b) abstracción con representación parcial. 65

Figura 5.4 Doble etiquetado hecho por el modelo de clasificación. 66

INDICE DE TABLAS.

Tabla 4.1 Resultados generales de pruebas de abstracción. 51

Tabla 5.1 Resultados de entrenamiento con más variaciones de tamaño de batch. 63

Tabla 5.2 Resultados de las pruebas con la segunda versión del dataset. 64

INDICE DE GRÁFICAS

Gráfica 5.1 Tiempo de entrenamiento total por prueba. 61
Gráfica 5.2 Tiempo de entrenamiento promedio por época en cada prueba. 62
Gráfica 5.3 Porcentaje de etiquetado de imágenes por prueba. 63

CAPÍTULO I. INTRODUCCIÓN.

1.1 Introducción.

Durante la mayor parte del siglo XX, la actividad humana se ha enfocado a la producción en masa de bienes comerciales, lo que ha sido parte de una revolución industrial que avanza a una velocidad impresionante. Ahora mismo estamos en un punto histórico en el que la industria se enfrenta a una nueva revolución, el uso del capital humano como principal fuerza de trabajo se está cambiando lentamente a un entorno más automatizado. Las máquinas de manufactura han ganado cada vez más terreno como la forma más segura y eficiente de producir bienes de consumo, por lo que el siguiente paso lógico: poner el funcionamiento de dicha maquinaria a cargo de agentes autónomos, o sea, sin necesidad de operadores humanos para realizar un trabajo determinado.

En las últimas décadas se ha impulsado el desarrollo de agentes robóticos para su implementación en las líneas de producción. Con avances tan claros como el uso de brazos robóticos con varios grados de libertad y la implementación de mesas de trabajo automatizadas, la industria 4.0 promete ser el siguiente gran paso en el desarrollo de la producción comercial. Como se menciona en el trabajo de (Larsen, Kim, Kupke, & Schuster, 2017) “las instalaciones de industria 4.0 deben ser flexibles y fáciles de reconfigurar. [...] las celdas robóticas de última generación son programadas para una tarea en específico que claramente define todos los pasos necesarios para llevarlas a cabo y solo puede reaccionar a pequeñas variaciones en el proceso”. Debido a esto, es importante denotar que aún falta un paso muy importante para conseguir la automatización total en las líneas de producción: la implementación de agentes inteligentes en estos robots de proceso.

Con la implementación de agentes inteligentes dentro de las plantas de producción vienen muchos retos y obstáculos a enfrentar, principalmente el hecho de mantener los sistemas autónomos consistentes y seguros. Por tal motivo una de las aplicaciones que requiere más trabajo al momento de implementase es la de los vehículos autónomos. Según (Zhao, Liang, & Chen, 2018) “un vehículo autónomo, también llamado robot móvil sobre ruedas, son un tipo de agente inteligente que llega a su destino basándose en la información obtenida a partir de

sensores, incluyendo la percepción del camino planificado, información de la ruta y los controles del vehículo". Con la definición previamente presentada se puede intuir que la cantidad de procesos necesarios para lograr el funcionamiento óptimo de uno de estos vehículos es bastante elevada, y que necesita mantener un tiempo de respuesta constante y eficiente.

Una solución para esto es la presentada en el trabajo de (Li, Adriaansen, Udding, & Progromsky, 2011) donde se describe a un AGV o "vehículo con guía automatizada", en el que se necesita tener una red de caminos y un control casi total de la localización relativa del vehículo en todo momento para asegurar su operación. Estas condiciones limitan de manera significativa el nivel de independencia de un vehículo de estas características, por lo que un nuevo enfoque es deseable para lograr las metas de la Industria 4.0.

1.2 Planteamiento del problema.

Dentro de la industria manufacturera es común que se generen retrasos ocasionados principalmente por errores humanos, ya sea por falta de atención de los operadores, o confusiones con las requisiciones de partes en las líneas de ensamblaje. Con el auge del uso de técnicas de inteligencia artificial es inevitable pensar en opciones para mejorar un proceso industrial eliminando el factor de error humano en el mayor grado posible.

La mayoría de los trabajos aplicados en la industria, en el contexto de vehículos no pilotados, se refieren a pequeños montacargas seguidores de línea como se muestra en la **¡Error! No se encuentra el origen de la referencia.**.

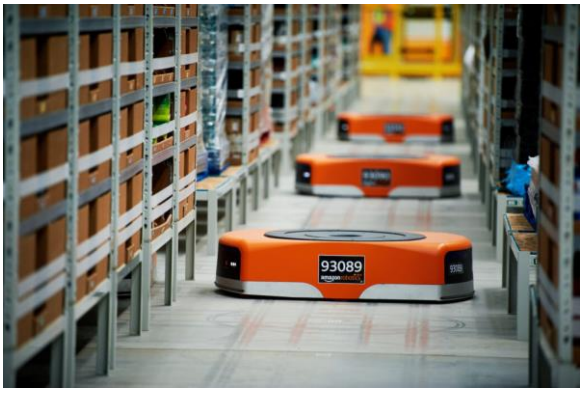


Figura 1.1 Vehículos autónomos guiados utilizados por Amazon.
Obtenido de: (Sutherland, 2018)

Sin embargo, la tecnología que hace posible el movimiento totalmente autónomo de un vehículo no ha sido utilizada de manera contundente dentro de una planta de producción. Una de las razones para que este tipo de vehículo no se haya implementado es la gran complejidad de sistemas involucrados en la operación y control de los agentes autónomos como se muestra en la Figura 1.2, en este trabajo se aborda uno de dichos sistemas: la localización del vehículo en el espacio físico.

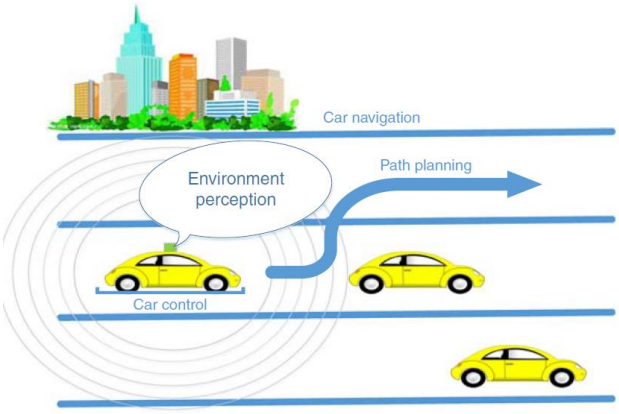


Figura 1.2 Diagrama de sistemas ligados a un vehículo autónomo.
Obtenido en: (Zhao, Liang, & Chen, 2018)

1.3 Alcances y limitaciones.

Como se mencionó anteriormente, la complejidad de cada sistema dentro de un vehículo autónomo es considerable, por lo que en este trabajo se abordará la parte consistente en el sistema de percepción de entorno, específicamente al que conlleva la identificación de líneas de pasillo desde el punto de vista del robot móvil, sin la parte de control de movimiento del robot.

1.4 Justificación.

Dentro del marco de la producción industrial, existe una problemática constante a enfrentar en el transporte de material de bodegas o almacenes hasta la línea de producción en donde se va a utilizar. Entre retrasos de órdenes de reposición de material, confusión por parte del operador del montacargas que lleva las piezas al no llegar a la estación de trabajo que las requería, o un

error al momento de pedir una reposición de material. Los tiempos muertos entre estas entregas pueden causar grandes cuellos de botella al momento de llevar a cabo el lote de producción.

1.5 Objetivos.

1.5.1 Objetivo general.

Desarrollar un sistema de visión computacional apoyado en algoritmos de procesado de imágenes y redes neuronales convolucionales con el fin de obtener información sobre la posición de un vehículo autónomo con respecto a las líneas de delimitación de pasillo en pisos de producción industrial.

1.5.2 Objetivos específicos.

- Contar con un programa de procesado de imágenes mediante Transformada de Hough para determinar condiciones específicas donde esta técnica no pueda obtener resultados.
- Obtener un *dataset* con los casos particulares de operación donde no se logre obtener un resultado con el algoritmo tradicional (Transformada de Hough).
- Contar con el etiquetado de imágenes en clases representativas para su identificación con una red neuronal convolucional.
- Obtener los valores de variables óptimos para el entrenamiento de la red neuronal.
- Contar con los parámetros idóneos para el funcionamiento de la red neuronal tras un proceso de experimentación y análisis de resultados.
- Obtener la integración de la técnica tradicional de procesamiento de imágenes con la aplicación de redes neuronales convolucionales para la solución del problema.

En el siguiente capítulo se abordarán algunos artículos relacionados con la problemática previamente descrita. Estos textos fueron recopilados a lo largo de la duración del proyecto con el objetivo de buscar aproximaciones reales aplicadas en la industria o en ambientes didácticos o académicos

CAPÍTULO II. ESTADO DEL ARTE.

Los artículos compilados en este capítulo provienen de fuentes variadas, principalmente de revistas y compendios científicos. Gracias a ello, podemos mostrar los avances actuales en algunas ramas de la automatización. Una de estas revistas es la *IEEE Transactions on Pattern Analysis and Machine Learning*, que como su nombre indica, se enfoca en la recolección de artículos sobre todas las áreas de visión computacional tradicional, así como ciertas técnicas de aprendizaje de máquina enfocado principalmente al reconocimiento de patrones. Otro ejemplo es el *IPOL Journal*, que se especializa en el rol de las matemáticas en el diseño de algoritmos propios del procesamiento y análisis de imágenes. Dentro de las fuentes utilizadas también se encuentra el *International Journal of Advanced Robotic Systems* que publica investigaciones de vanguardia en todos los campos asociados con robótica. Como última fuente listada tenemos la revista *Procedia Manufacture*, que nos aporta la perspectiva de la industria en las investigaciones realizadas donde ese utiliza tecnología de automatización.

Como se puede apreciar, los enfoques de estas revistas científicas varían en las técnicas y procedimientos, pero se juntan en la meta de recolectar información en el campo de la automatización tanto de procesos como de sistemas robóticos, siendo un punto en común muy importante en que la mayoría de estas y otras usadas en este trabajo son de acceso abierto.

En la siguiente subsección se da un pequeño resumen de los artículos analizados como referencia para este trabajo de tesis.

2.1 *Deep Hough transform for semantic line detection*.

En este artículo de (Zhao, Han, Zhang, Xu, & Cheng, 2021) podemos destacar el concepto de “línea semántica”, lo que describe a una línea que aporta contexto a una imagen, como puede ser el horizonte en una escena natural o el borde de un objeto prominente en una fotografía. En el procedimiento descrito se utilizan las características matemáticas que definen a las líneas como ventaja para poder destacarlas en la imagen, utilizando la transformación de Hough para parametrizar estas propiedades, en el que posteriormente se realiza la detección.

En el método propuesto en este artículo se extrae una representación a nivel píxel de la imagen a analizar con un codificador basado en una red neuronal convolucional. Al resultado se le realiza la transformación de Hough para convertir esta representación de características a un espacio paramétrico, lo que vuelve más eficiente la detección de picos en los datos transformados.

2.2 *Iterative Hough Transform for line detection in 3D point clouds*.

Dentro de este trabajo realizado por (Dalitz, Schramke, & Jeltsch, 2017), se implementa la transformación de Hough con un objetivo muy particular. Teniendo conjuntos de datos guardados en 3 dimensiones, se busca utilizar un sistema de votaciones en los resultados del algoritmo de la transformación de Hough para poder identificar para cada punto la línea detectada en la que pertenezca, con lo que se puede utilizar para la clasificación de datos o para el trabajo con conjuntos de datos entregados por ciertos sensores, como lo son los LIDAR.

2.3 *New approach for beacons based mobile robots localization using Kalman filters*.

En los últimos años se han generado estrategias muy diversas con el fin de dotar a un vehículo la información de su posición en el entorno donde opera. Estas estrategias van desde cintas magnéticas hasta marcas físicas colocadas para triangular la posición del vehículo. Sin embargo, la industria se mueve hacia la operación de vehículo en ambientes donde haya obstáculos desconocidos para el sistema.

En este trabajo de (Moreira, Costa, & Lima, 2021), se implementa un sistema de posicionamiento redundante un sensor LIDAR YDLIDAR X4 montado en el robot, marcadores físicos en el en el entorno, así como un sistema de cámaras para captura de movimiento. Para lograr calcular la posición del robot se utiliza un método conocido como “Filtro extendido de Kalman”, que toma como base los datos obtenidos por el LIDAR para obtener una ecuación no lineal que estima el movimiento del robot dentro de su espacio de coordenadas físicas, corroborando este resultado con la posición absoluta obtenida con el sistema de captura de movimiento.

2.4 *Power line detection using Hough transform and line tracing techniques*.

Dentro del trabajo hecho por (Barker, Mills, & Langlotz, 2016) se detalla la problemática de detectar automáticamente la presencia de líneas de alta tensión en imágenes tomadas desde perspectivas a distancia media o lejana, con el fin de monitorear su estado para mantenimiento preventivo evitando la inspección cercana a las líneas.

Dado que las líneas de energía no se encuentran totalmente rectas debido a su peso, es necesario implementar un algoritmo capaz de detectar ese tipo de líneas curvas, lo que se puede lograr modificando la operación de la transformada de Hough.

Como primer paso para el algoritmo, las imágenes de las líneas deben contener la totalidad del cable de alta tensión, lo que asegura captar la curvatura total de la línea. Además, antes de proceder se debe transformar la imagen para que el cable esté alineado con el plano de la cámara. La primera parte del algoritmo es realizar sobre la imagen en cuestión un proceso de filtrado donde se realizan las siguientes modificaciones:

- Difuminado Gaussiano con un núcleo de 5x3 pixeles, un valor mayor en el eje Y, lo que da por resultado una mayor difuminación en los bordes verticales de los objetos.
- Usando promediado no local en la imagen, se elimina el ruido de entre los objetos presentes.
- Se realiza un umbral de binarización para diferenciar completamente a los objetos presentes en la imagen del fondo.
- Se aplica un adelgazamiento de bordes, para hacer más eficiente a la transformación de Hough.

Una vez teniendo este filtrado en la imagen a trabajar, el algoritmo de Hough se modifica usando la ecuación para buscar los componentes de una parábola en vez de los de una línea recta, añadiendo además un muestreo aleatorio en vez del muestreo por fuerza bruta que utiliza por defecto.

2.5 *Autonomous navigation of mobile robots in factory enviroments*.

Dentro de las capacidades anteriormente mencionadas de un vehículo autónomo está el encontrar el camino más apto para llegar a su destino. En este trabajo de (Harapanahalli, y otros, 2019) se abordan diferentes tecnologías que permiten generar algoritmos específicos lo suficientemente robustos para mantener al vehículo en movimiento sin producir accidentes entre los demás ocupantes de su espacio de trabajo.

En primer lugar, tenemos a los sensores para monitorear la posición del robot, que se pueden dividir en dos clases: sensores externos y sensores internos. Los primeros se utilizan para medir la velocidad de movimiento o el avance de las ruedas del robot para estimar su posición y orientación desde un punto inicial, mientras que los sensores externos utilizan tecnologías como balizas de radiofrecuencia, marcas visibles en el entorno o señales como WiFi o Bluetooth.

Parte de la problemática yace en poder crear un mapa del entorno del robot de manera simultánea con la localización de la misma unidad, lo que se logra combinando la información obtenida por estos sensores con imágenes ingresadas al sistema por cámaras (ya sean tradicionales o con mapeo de profundidad) para aportar más información sobre el entorno.

Se propone entonces tomar esta información y procesarla con métodos basados en redes neuronales para identificar objetos específicos en el entorno, lo que permite realizar un mapeado más preciso. Así mismo, el trabajo también propone tener dos planificadores: uno global para considerar a los obstáculos fijos y otro local, con la finalidad de monitorear el entorno cercano constantemente en busca de obstáculos dinámicos.

2.6 *Comparison of three key remote sensing technologies for mobile robots locations in nuclear facilities.*

Dependiendo del ambiente en que se esté planificando la implementación de

tecnologías de automatización, sus requerimientos mínimos tienden a variar de manera considerable. Un ejemplo de esto se presenta en este trabajo de (Jonasson, Vale, & Ramos Pinto, 2021), donde se plantea la operación de un vehículo autónomo en las instalaciones de un reactor nuclear.

La investigación prueba la operación de tres tipos de sensores:

- Cámaras de color-profundidad RGB-D.
- Sensores LIDAR.
- Radar de onda milimétrica.

Dentro de las instalaciones nucleares existen condiciones de operación mucho más exigentes que en otro tipo de industrias como lo son los niveles altos de radiación, contaminación ambiental en forma de polvo, campos magnéticos residuales, condiciones desfavorables de iluminación y la restricción de entrada de personal humano a ciertos sectores de las instalaciones por motivos de seguridad.

Para probar la efectividad de los sensores se colocaron uno encima de otro, como se muestra en la Figura 2.1, usando una cámara RGB-D Intel *Realsense* d435, un sensor LIDAR *Celodyne* VLP-16, y un Radar *mmWave* Demo AWR 1443 BOOST montados en una cubierta de aluminio. Una vez montados los sensores se recorrió un pasillo de las instalaciones para recabar los datos obtenidos por cada sensor.

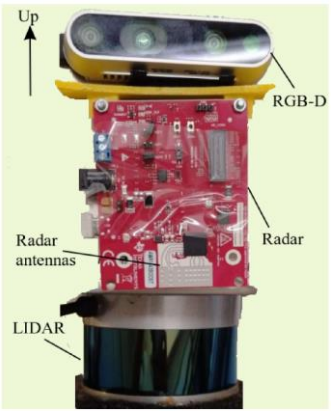


Figura 2.1 Distribución de los sensores para prueba de funcionamiento.
Obtenido de: (Jonasson, Vale, & Ramos Pinto, 2021)

Como resultado se tienen las nubes de puntos que generan los sensores, mostradas en la **¡Error! No se encuentra el origen de la referencia.**.

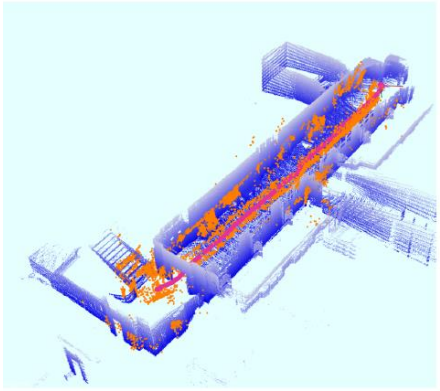


Figura 2.2 Mapa generado a partir de la información de los sensores.
Obtenido de: (Jonasson, Vale, & Ramos Pinto, 2021)

Lo que se destaca es como el radar funciona muy bien para detectar obstáculos, mientras que su densidad de puntos no aporta mucha información del entorno. En cambio, la cámara RGB-D aporta una densidad de puntos muy elevada, pero depende de niveles de iluminación estable para asegurar la calidad de las imágenes. Con esto último se puede concluir que para asegurar una operación robusta es ideal combinar varias tecnologías en el mapeo del entorno, siendo el principal objetivo de esto volver redundante la información de entrada en el vehículo, para que en caso de que uno de los sensores falle, se pueda continuar la operación con la información de los otros sensores

Tomando en cuenta toda esta información, se destacan las cualidades que debe tener este proyecto. En primera instancia como se menciona en los trabajos de (Zhao, Han, Zhang, Xu, & Cheng, 2021), (Dalitz, Schramke, & Jeltsch, 2017) y (Barker, Mills, & Langlotz, 2016) el uso de la transformada de Hough nos permite identificar de manera clara la presencia de líneas rectas en el ambiente observado, lo que nos interesa principalmente en este trabajo. A su vez, los trabajos de (Jonasson, Vale, & Ramos Pinto, 2021), (Harapanahalli, y otros, 2019) y (Moreira, Costa, & Lima, 2021) nos dejan la opción de tener como apoyo varias formas de detección para mantener un sistema lo suficientemente robusto para poder lograr la operación deseada. Con esta perspectiva se opta por el uso de las redes neuronales convolucionales, como en el trabajo de (Harapanahalli, y otros, 2019), para tener una alternativa de operación en caso de que el procesado de imágenes por transformada de Hough no cuente con información del entorno

suficiente para tomar una decisión acertada. A su vez, esto nos evita la necesidad de tener sensores externos al vehículo, volviendo su operación más flexible y fácil de alcanzar

CAPÍTULO III. MARCO TEÓRICO.

Como se mencionó en el capítulo anterior, parte del objetivo de este trabajo es lograr procesar imágenes del mundo real para obtener información del entorno de operación de un vehículo autónomo, lo que no es posible lograr sin utilizar técnicas y librerías propias de la rama de inteligencia artificial conocida como visión artificial, también llamada visión por computadora.

3.1 Visión artificial.

Según el libro "Procesamiento digital de imágenes" (Chacón M., 2007), la visión artificial es la capacidad de la máquina de ver el mundo que la rodea, más precisamente, deducir la estructura y las propiedades de un mundo tridimensional a partir de imágenes bidimensionales. Con esta definición en mente podemos empezar a vislumbrar las aplicaciones a las que podemos llegar, dependiendo de lo que se busque obtener de la imagen analizada, que para propósitos de este proyecto es obtener información de entorno en forma de los límites de un pasillo.

Al tener que detectar líneas rectas en estos límites de pasillos, se tiene que buscar una técnica que nos permita obtener las características de una línea recta a partir de una imagen del entorno, lo que se compagina con el funcionamiento de la transformación de Hough. Este proceso definido en 1962 por Paul Hough (Hough & Arbor, 1962) se utiliza para encontrar patrones complejos de líneas en espacios pictóricos, como implemento para identificar el recorrido de partículas subatómicas en un campo de visión. A continuación, se describe brevemente las operaciones necesarias para lograr la operación de procesado de imágenes por transformada de Hough.

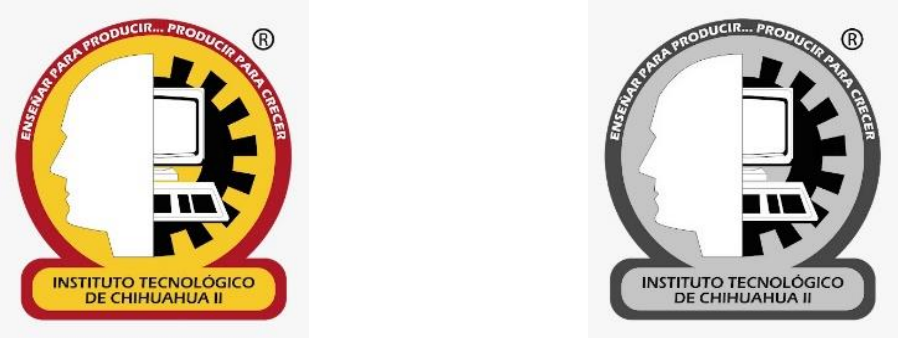
3.1.1 Escala de grises.

Según la página de documentación oficial de *OpenCV* (Open Source Computer Vision, 2023) la conversión entre canales de color y una salida en escala de grises se realiza sumando las luminosidades de los tres canales de color presentes en el modelo RGB (Rojo, Verde y Azul por sus iniciales en inglés).

Esta operación se describe en la ecuación (3.*1*).

$$Y = R * 0.299 + G * 0.587 + B * 0.114 \quad (3.1)$$

Dando como resultado una salida como en la Figura 3.1.



a) b)

Figura 3.1 a) Imagen original y b) Imágen convertida a escala de grises.
Obtenido de: Elaboración propia.

3.1.2 Difuminado.

OpenCV (Open source computer Vision, 2023) también implementa una función de difuminación o *Blur* con el fin de suavizar bordes de objetos poco importantes en la imagen. En la documentación oficial podemos apreciar que la función aplica un núcleo o *kernel* normalizado *K*, tomando la altura *h* y la anchura *w* de este, como se ve en la ecuación (3.*2*).

$$K = \frac{1}{h*w}\begin{pmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{pmatrix} \quad (3.2)$$

Esto para dar como resultado una imagen como se muestra en la Figura 3.2.

Figura 3.2 Resultado de la operación de Difuminado.
Obtenido en: Elaboración propia.

Las siguientes definiciones de operaciones fueron obtenidas del libro "Procesamiento digital de imágenes" (Chacón M., 2007), "Visión por computador: imágenes digitales y aplicaciones" (Pajares Martinsanz & de la Cruz García, 2008) y "Tratamiento digital de imágenes" (González & Woods, 1996), con el fin de contrastar definiciones y dar una mejor descripción de estas.

3.1.3 Binarización.

El uso de umbrales en el tratamiento de imágenes constituye una de las principales técnicas en los sistemas de visión industrial para la detección de objetos, especialmente en aplicaciones que requieran procesar una cantidad elevada de datos.

Teniendo el caso de una imagen con objetos claros sobre un fondo oscuro con 2 tonos dominantes, una forma de separar los objetos del fondo es seleccionar un nivel T que separe esos dos tonos de intensidad, tal que si un pixel (x,y) para el cual $f(x,y) > T$ sea perteneciente a un objeto, mientras que en el caso contrario sea del entorno.

Con esta definición, se puede formular una función para T que quedaría como muestra la ecuación (3.3:

$$T[x, y, p(x, y), f(x, y)] \tag{3.3}$$

Donde *f(x,y)* es la intensidad en el punto *(x,y)* y *p(x,y)* puede ser alguna propiedad local del vecindario de pixeles centrado en *(x,y)*.

Al pasar por esta función *T* definimos el umbral para crear la imagen *g(x,y)*, que se caracteriza por la ecuación (3.*4*:

$$g(x,y) = \begin{cases} 0\ si\ \ f(x,y) > T \\ 1\ si\ f(x,y) \leq T \end{cases} \tag{3.4}$$

Lo que nos dice que a los pixeles con valor de cero son parte del objeto y los que tengan un 1 son parte del entorno. Se puede aplicar una binarización inversa, lo que cambia la definición de g(x,y) como muestra la ecuación(3.5:

$$g(x,y) = \begin{cases} 0\ si\ \ f(x,y) \leq T \\ 1\ si\ f(x,y) > T \end{cases} \tag{3.5}$$

Este simple cambio hace que los pixeles del fondo sean negros y los de los objetos sean blancos.

Existen varias formas de elegir el valor del umbral óptimo para detectar los objetos presentes en una imagen, y una de las más utilizadas es el Método de Otsu. En este proceso se toma en cuenta que los tonos de la imagen se distribuyan en dos densidades de probabilidad gaussiana. A medida que las curvas gaussianas se aproximen a las mostradas en el histograma real las desviaciones estándar disminuyen, lo que nos deja la opción de elegir el umbral que minimice la suma de varianzas en ambas curvas del histograma. Esto último se representa con las ecuaciones (3.*6* y (3.*7*, donde se tiene como *L* a los niveles de intensidad, *T* representa al umbral buscado y sus probabilidades acumuladas hasta *T*, y desde *T* hasta *L*, se representa con w_1 y w_2.

$$w_1(t) = \sum_{z=1}^{T} P(z) \tag{3.6}$$

$$w_2(t) = \sum_{z=T+1}^{L} P(z) \tag{3.7}$$

Teniendo los valores w_1 y w_2, se obtienen la media y la varianza asociada a cada uno

$$\mu_1(t) = \sum_{z=1}^{T} zP(z) \tag{3.8}$$

$$\mu_2(t) = \sum_{z=T+1}^{L} zP(z) \tag{3.9}$$

$${\sigma_1}^2(t) = \sum_{z=T+1}^{L} \left(z - \mu_1(t)\right)^2 \frac{P(z)}{w_1(t)} \tag{3.10}$$

$${\sigma_2}^2(t) = \sum_{z=T+1}^{L} \left(z - \mu_2(t)\right)^2 \frac{P(z)}{w_2(t)} \tag{3.11}$$

Finalmente, se obtiene la varianza ponderada:

$${\sigma_w}^2(t) = w_1(t){\sigma_1}^2(t) + w_2(t){\sigma_2}^2(t) \tag{3.12}$$

Con este modelo, el umbral *T* que nos de la menor varianza ponderada es el que se utiliza para binarizar la imagen. En la **¡Error! No se encuentra el origen de la referencia.** se muestran los tres tipos de binarización detallados en esta sección.

Figura 3.3 a) Binarización estándar, b) Binarización inversa y c) Binarización Otsu.
Obtenido en: Elaboración propia.

3.1.4 Operaciones morfológicas.

Para trabajar con las imágenes, es necesario reducir los detalles innecesarios de la misma o resaltar la información que contiene dicha imagen. Algunas de las operaciones que son necesarias para esta manipulación de la información son las transformaciones morfológicas, principalmente la dilatación, la erosión, la apertura y el cierre, que se describen a continuación.

3.1.4.1 Dilatación.

La transformación morfológica de la dilatación combina dos conjuntos por adición de vectores, la dilatación X⊕B representa todas las adiciones vectoriales entre los elementos de X y B, expresándose como se muestra en la ecuación (3.*13*).

$$X \oplus B = \{d \in E^2 : d = x + b \; para \; cada \; x \in X \; y \; b \in B\} \qquad (3.13)$$

En este caso, el elemento *X* representa los pixeles de la imagen, mientras el elemento *B* es un componente estructural que denota cuantos pixeles se va a expandir la imagen original. Gráficamente, el proceso para completar la operación de dilatación conlleva los siguientes pasos: primeramente, se recorre el arreglo de pixeles de izquierda a derecha y de arriba abajo. Por cada incremento de pixeles se posiciona el componente estructural donde se encuentre el valor de 1 (uno) en el valor del píxel. Una vez posicionado el origen del elemento estructural se hace una unión entre los pixeles del elemento estructural y la sección de la imagen donde se posicionó. Si los pixeles de la imagen que contienen unos coinciden con los unos del elemento estructural, el origen de dicho elemento es marcado como el valor de 1 (uno).

Esta operación da como resultado una expansión en las líneas pixeles en la imagen a la que se aplica, como podemos ver en la Figura 3.4.

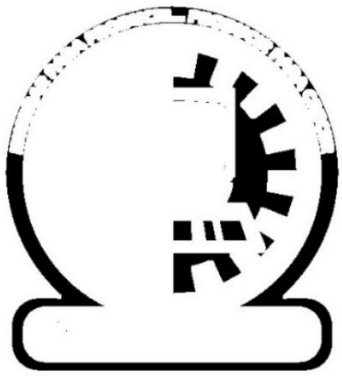


Figura 3.4 Resultado de la operación de Dilatación.
Obtenido en: Elaboración propia.

3.1.4.2 Erosión.

La erosión es una operación de substracción de vectores, por lo que se puede decir que es la contraparte de la dilatación, como lo define la ecuación (3.14).

$$X \ominus B = \{d \in E^2 : d = x + b \; para \; cada \; x \in X \; y \; b \in B\} \qquad (3.14)$$

Según esta definición, para cada punto d del conjunto X que es probado, el resultado de la erosión representa a los puntos d en donde todos los posibles resultados de $d + b$ están en X.

En aplicaciones gráficas la erosión comienza recorriendo la imagen, como ejemplo, de arriba hacia abajo y de izquierda a derecha. Donde se encuentre el valor de 1 (uno) se posiciona el elemento estructural B. Si los unos en esa sección de la imagen coinciden con los unos del elemento estructural se marca el píxel donde se encuentra el origen del elemento estructural como "1".

El resultado de la operación se muestra en la Figura 3.5. Se observa cómo se eliminan detalles del contorno de la imagen original, por lo que a la erosión también se le llama "reducción".

Figura 3.5 Resultado de la operación de Erosión.
Obtenido en: Elaboración propia.

3.1.4.3 Clausura y apertura.

Las operaciones morfológicas como la erosión y la dilatación son no invertibles, lo que significa que no podemos devolver los valores originales a los pixeles una vez aplicadas las operaciones.

Cuando se aplica una erosión seguida de una dilatación se le llama “Transformación de Apertura”, o apertura a secas. La apertura de X con respecto al elemento estructural B se define por la ecuación (3.*15*).

$$X \circ B = (X \ominus B) \oplus B \tag{3.15}$$

Ahora bien, si se realiza una dilatación seguida de una erosión se conoce como “Transformación de cierre” o cierre y se representa como muestra la ecuación (3.16).

$$X \cdot B = (X \oplus B) \ominus B \tag{3.16}$$

La apertura y el cierre son operaciones que se emplean con el objetivo de eliminar detalles de la imagen original sin perder los contornos generales de los objetos contenidos en ella, mientras dichos detalles sean más pequeños que el elemento estructural utilizado para realizar la operación.

En la Figura 3.6 se muestra cómo se modifica la imagen base (a) con una apertura (b) y un cierre (c).

a) b) c)

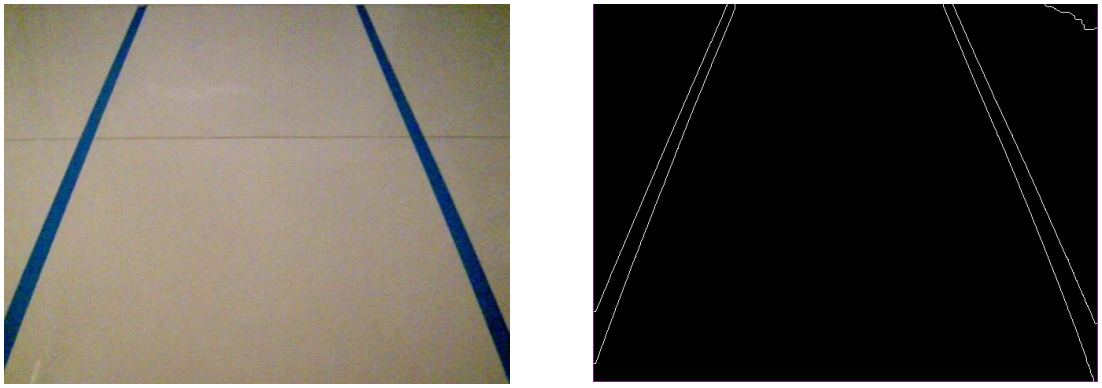
Figura 3.6 a) Imagen base, b) Apertura y c) Clausura.
Obtenido en: elaboración propia.

3.1.5 Función *Canny*.

Una vez obtenida la imagen binaria y que sus características han sido resaltadas por medio de las operaciones morfológicas podemos obtener el contorno de los objetos presentes en la imagen. Para ello existen varios algoritmos, entre ellos el propuesto por Canny en 1986.

Para hacer funcionar este algoritmo se definen tres condiciones de operación. Las primeras dos son las siguientes:

La primera es el error de detección, siendo que se tiene que asegurar que no se detecten bordes inexistentes y que se identifiquen todos los bordes presentes en la imagen. La segunda nos describe la distancia entre el píxel de borde detectado y el borde real, la cual debe ser la mínima posible. Por último, la tercera condición conlleva asegurar que no se detecten los mismos bordes varias veces, para no caer en redundancias innecesarias. En la Figura 3.7 se puede ver una imagen base (a) y el resultado de aplicarle la función *Canny* (b).



a) b)

Figura 3.7 a) Imagen base y b) Resultado de Canny()
Obtenido en: Elaboración propia.

3.1.6 Transformada de Hough.

La transformada de Hough localiza diferentes formas en una imagen base tales como líneas y curvas. Para este proyecto centraremos la atención en la localización de líneas rectas.

Tomando como punto de partida un píxel de coordenadas (x_i, y_i) y que la ecuación de la recta tiene la forma:

$$y_i = ax_i + b \tag{3.17}$$

Bajo estas definiciones, si variamos los valores de a y b se pueden obtener una infinidad de líneas rectas que pasen por la coordenada (x_i, y_i).

Se puede cambiar la ecuación para tomar como parámetros a *a* y *b* dejando la ecuación (3.18 como se muestra a continuación.

$$b = -ax_i + y \tag{3.18}$$

Lo que nos deja ahora un plano *ab*, donde se puede crear una línea concreta para el punto (x_i, y_i). Si consideramos otro punto (x_j, y_j) podemos generar una línea que intercepte a la línea de (x_i, y_i) en el punto (a', b') del plano *ab*, lo que nos permite mapear las coordenadas con la forma (x, y) como rectas del plano *ab*.

Existe un problema al generar este mapeo, debido a que la pendiente y la intersección de las líneas tienen a infinito a medida que la línea llega a una posición vertical. Una forma de corregir el error es parametrizar la recta como se muestra en la en la ecuación (3.19.

$$\rho = x\cos\theta + y\sin\theta \tag{3.19}$$

Con esta nueva parametrización, en vez de que los valores (x_i, y_i) generen líneas, los parámetros $\rho\theta$ generan ondas sinusoidales. En esta forma de representar los puntos, el parámetro ρ corresponde a la distancia entre el punto (x_i, y_i) y el origen de la imagen, mientras que θ es el ángulo entre la perpendicular del origen al punto y la recta que pasa por (x_i, y_i).

En este nuevo plano $\rho\theta$, los puntos (x, y) generan su respectiva sinusoidal. Al momento de que varios puntos (x, y) se alineen sus sinusoidales van a interceptarse. Una vez terminado el cálculo de estas sinusoidales el algoritmo realiza un proceso de votación: de entre todos los puntos calculados de las sinusoidales se da un voto cada vez que hay una intersección, siendo identificada la línea que tenga más votos.

En Figura 3.8, se muestra el resultado de correr el algoritmo de transformada de Hough en una imagen con líneas presentes, marcando las líneas detectadas en color verde.

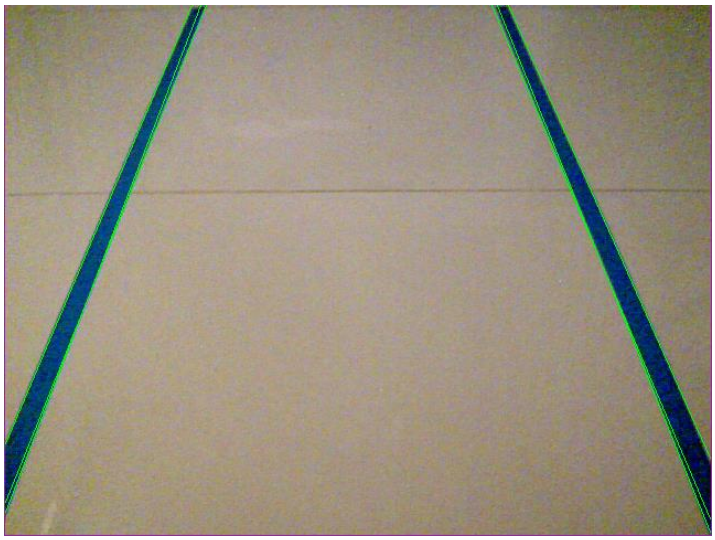


Figura 3.8 Líneas representadas por la transformada de Hough.
Obtenido en: Elaboración propia.

Teniendo como base el sistema de procesado de imágenes, se pueden presentar casos específicos en los que no sea suficiente tener las líneas rectas presentes en una sola toma de imagen, como se ilustra en la Figura 3.9, tal que el algoritmo de procesamiento y abstracción no es suficiente para asegurar la identificación del pasillo.

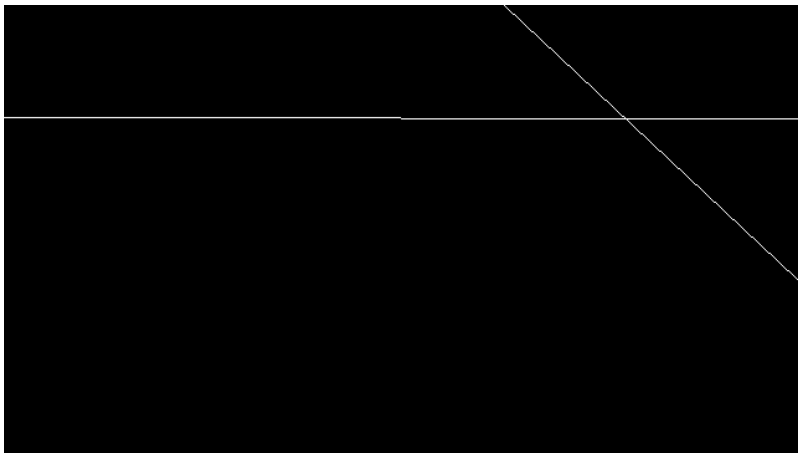


Figura 3.9 Caso de ambigüedad detectado en la abstracción.
Obtenido en: Elaboración propia.

Una posible solución para esto es la implementación de un sistema de inferencia, en el que podamos clasificar los casos detectados por las líneas de Hough, de tal manera en que se tenga

más información necesaria para tomar una decisión sobre la ubicación del vehículo sobre el pasillo.

Como se menciona en el capítulo IV de este trabajo, la manera más flexible de implementar un sistema de inferencia capaz de identificar diferentes casos de operación son las redes neuronales. A continuación, se repasa su funcionamiento.

3.2 Redes neuronales.

3.2.1 Descripción general.

Las redes neuronales artificiales surgieron como una forma de emular el funcionamiento de las neuronas como elementos de procesamiento de información. Una neurona biológica está formada por tres elementos básicos: las dendritas, el núcleo o soma y el axón, que se ilustran en la Figura 3.10.

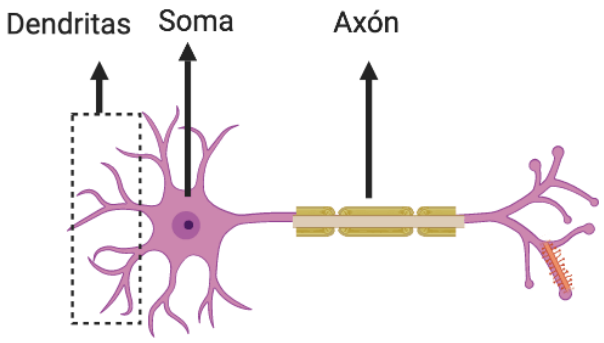


Figura 3.10 Diagrama de una neurona biológica.
Obtenido en: (García, 2019)

Las dendritas son las responsables de captar señales eléctricas provenientes de otras neuronas. El núcleo o soma toma las señales eléctricas captadas por las dendritas y las procesa químicamente, si se genera la suficiente energía interna, la neurona se dispara. Una vez disparada, la nueva señal eléctrica pasa por el axón y sale por otro juego de dendritas.

Al dispararse la señal eléctrica, puede ocurrir el proceso de sinapsis, donde la información obtenida por una neurona pasa a la siguiente al generar una unión entre sus dendritas, lo que genera la memoria y el aprendizaje en el cerebro. La red neuronal, como su nombre lo indica,

es la estructura que se crea al interconectar varias neuronas entre sí, lo que provee al sistema la propiedad del procesamiento paralelo entre sus unidades.

Las redes neuronales artificiales, o RNA por sus siglas, es la recreación de este proceso en un modelo matemático representado en la Figura 3.11.

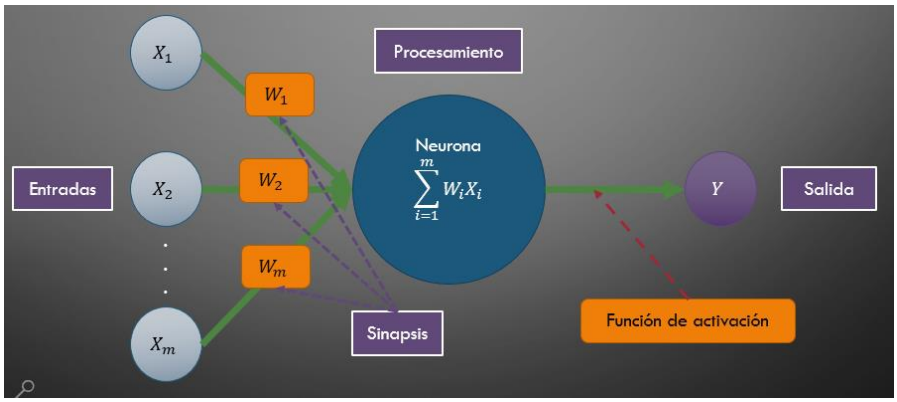


Figura 3.11 Diagrama de una neurona artificial.
Obtenido en: elaboración propia.

En este caso, se representa a las dendritas con un vector de entradas *x*, que es multiplicado por un peso un peso sináptico *w*. Este *w* es lo que genera el aprendizaje en la red mediante un proceso de entrenamiento que se explica más adelante en esta sección.

Los datos de entrada del vector *x*, una vez multiplicados por su respectivo peso *w* se suman, siendo este un símil del proceso químico del núcleo de la neurona biológica. Una vez obtenido el resultado de la sumatoria, se introduce en una función de activación, denotada por la ecuación (3.*20*.

$$y(n) = f(\sum_{i=1}^{n} x_i(n) w_i\,(n) \pm b) \qquad (3.20)$$

Esto indica que información pasará como salida de la neurona artificial. Existen varias funciones de activación y estas pueden ser lineales o no lineales. Dependiendo del dominio del

problema se utilizará alguna de ellas en concreto. Entre las funciones de activación podemos mencionar las 4 más comunes:

Función lineal: Entrega el valor obtenido por la neurona de manera directa, sin modificaciones.

$$f(x) = x \tag{3.21}$$

Función escalonada: Entrega un 1 si el valor obtenido por la neurona es mayor a 0, y entrega 0 si dicho valor es igual o menor a 0.

$$f(x) = \begin{cases} 1 \; si \; x \geq 0 \\ 0 \; si \; x < 0 \end{cases} \tag{3.22}$$

Función sigmoide: Entrega el valor obtenido por la neurona escalado en un rango entre 0 y 1.

$$f(x) = \frac{1}{1-e^{-x}} \tag{3.23}$$

Función rectificador lineal unitario (ReLU): Entrega el valor obtenido por la neurona sin modificaciones si es mayor que 0, en caso contrario entrega 0.

$$f(x) = \begin{cases} x \; si \; x \geq 0 \\ 0 \; si \; x < 0 \end{cases} \tag{3.24}$$

La representación gráfica de las funciones antes presentadas se puede visualizar en la Figura 3.12.

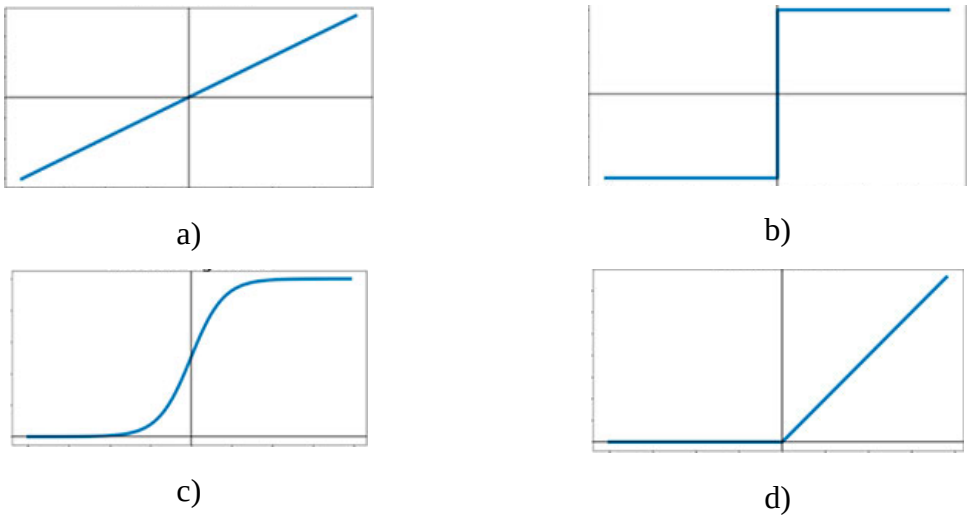


Figura 3.12 Formas de funciones de activación. a) Lineal, b) Escalonada, c) Sigmoide y d) ReLU. Obtenido en: (Alonso , 2021)

3.2.2 Entrenamiento de una red neuronal.

Las RNA implementan un proceso de aprendizaje basado en la modificación de los pesos sinápticos. Existen dos formas de aprendizaje: el supervisado y el no supervisado. Para este trabajo nos centraremos en el primero.

Como primer paso se debe separar el conjunto de datos o *dataset* en dos conjuntos para llevar a cabo el aprendizaje, uno de ellos servirá de material de entrenamiento mientras que el otro será el conjunto de pruebas para medir la efectividad del modelo entrenado. Lo que se busca con un aprendizaje supervisado es lograr ajustar las salidas de la red para que coincidan con las clasificaciones previamente definidas para los datos de entrada, de modo que una vez que se introduzcan nuevos datos se logre una clasificación acertada tomando en cuenta los grupos previamente presentados a la RNA. Por propósitos explicativos el siguiente ejemplo corresponde al modelo de una sola neurona o perceptrón desarrollado por Frank Rosenblant.

Para saber si una neurona obtiene una salida correcta o incorrecta se define una regla de aprendizaje, que para este caso de ejemplo es que, si la salida y la entrada de la neurona se activan, la conexión entre ambas se fortalece, lo que se representa aumentando el peso sináptico de la entrada en cuestión.

Como datos de entrada al modelo tenemos los pares (X_k, O_T), siendo X_k el dato de entrada y O_T la salida esperada para dicha entrada. Cuando se termina el cálculo de la salida O_{pk} para la entrada X_k, se compara con la salida esperada O_T. La diferencia entre estas salidas (O_{pk} y O_T) define el error de salida, que se utiliza para modificar los pesos sinápticos w.

El proceso se puede resumir en los siguientes pasos:

1) Los pesos *w* se inicializan con valores aleatorios pequeños.
2) Se calcula

$$net = \sum_{k=1}^{n} x_k(n)\, w_k(n) + b \tag{3.25}$$

3) Se implementa la función de activación escalonada, tal que

$$O_{pk} = \begin{cases} 1 \; si \; net > 0 \\ 0 \; si \; net \leq 0 \end{cases} \tag{3.26}$$

4) Calcular el error entre O_{pk} y O_T

$$\nabla = O_T - O_{pk} \tag{3.27}$$

5) Se actualizan los pesos *w* y el sesgo *b*

$$w_j(n+1) = w_j(n) + \nabla x_j \tag{3.28}$$

$$b(n+1) = b_j(n) + \nabla \tag{3.29}$$

Siendo *n* el número de iteración actual.

6) Repita los pasos del dos al cinco hasta que no se registre un cambio en los valores de los pesos *w* o que el máximo número de iteraciones sea alcanzado.

Este proceso funciona para casos donde se requiera una clasificación lineal, dicho de otro modo, una clasificación donde los datos solo se separen en 2 grupos por medio de una línea divisoria.

Para casos más complejos ya se habla de redes de neuronas con varias capas que pueden lograr clasificaciones más complejas. En estos casos se modifica la forma en que calculamos el error buscando ahora un error cuadrático, como se muestra en la ecuación (3.30.

$$E = \frac{1}{2}\left(O_T - O_{pk}\right)^2 \tag{3.30}$$

Esta modificación convierte la búsqueda del error más pequeño en un problema de mínimos cuadrados, lo que nos permite resolverlo buscando el valor negativo del gradiente, actualizando los pesos como se muestra en la ecuación (3.31.

$$W(n+1) = W(n) - \alpha \frac{\partial E(W)}{\partial W} \tag{3.31}$$

Donde α es el parámetro de la regla de aprendizaje. Este gradiente se puede expresar también en términos de las salidas esperada y calculada

$$\frac{\partial E(W)}{\partial W} = -(O_T - WX)X \tag{3.32}$$

Lo que nos permite reescribir la ecuación (3.33 como

$$W(n+1) = W(n) + \alpha(O_T - WX)X \tag{3.33}$$

3.2.3 Funcionamiento de redes neuronales convolucionales.

Tomando como base el funcionamiento de las RNA podemos hablar de un tipo más específico de estructura: las redes neuronales convolucionales o RNC.

Como se describe en el artículo de (O'shea & Nash, 2015), a diferencia más notable entre ambos tipos de redes es que las RNC son principalmente utilizadas para el reconocimiento de patrones con imágenes. Esto último hace que la arquitectura de la red se enfoca en organizarse de tal manera que pueda manejar un tipo de datos específico, en este caso, las imágenes. Para lograrlo, las neuronas de la RNC se organizan en tres dimensiones: las dimensiones espaciales de la imagen (ancho y alto) y la profundidad del volumen de activación.

Las RNC tienen tres tipos de capas. El primer tipo es el de convolución, luego tenemos las capas de *pooling* o agrupación y por último una capa densa o completamente conectada.

La capa de convolución es una parte vital para las RNC. Los parámetros de esta capa se enfocan en el uso de *kernels* o núcleos, que son usualmente de dimensiones pequeñas, pero se esparcen por toda la dimensión de las entradas de la red. Cada núcleo crea un mapa de activación correspondiente, lo que al multiplicarse por la región de la imagen da un producto escalar que resalta la característica buscada por dicho núcleo, lo que se ejemplifica en la Figura 3.13.

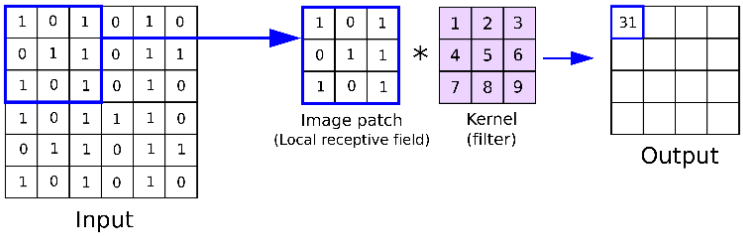


Figura 3.13 Diagrama de proceso de convolución.
Obtenido en: (Reynolds, 2019)

Este proceso reduce la complejidad del modelo implementando tres parámetros, que son los siguientes.

- La profundidad de volumen de salida, que representa a todas las neuronas de entrada que conectan con la primera capa de convolución. Al reducir la profundidad resulta en una red más compacta, con el costo agregado de perder también las capacidades de reconocimiento que esto conlleva.
- El paso o *stride*, que es el rango en el espacio dimensional de la entrada en que se va a mover nuestro núcleo. Si el paso tiene por valor el 1 se genera una sobreposición de núcleo, lo que deja una cantidad de activaciones muy elevada. Si se asigna un valor mayor al *stride*, podemos no solo evitar esta sobreposición, sino reducir significativamente el espacio dimensional de la salida.
- El *zero-padding* o relleno cero, es el proceso de rellenar los bordes de la imagen de entrada para controlar el tamaño dimensional de la misma.

Al combinar estos tres parámetros podemos alterar el espacio dimensional de nuestra salida en la capa de convolución.

La capa de agrupación o *pooling*, que como su nombre indica, junta aún más los mapas de activación del modelo para reducir su complejidad nuevamente. Esto es posible gracias a la función de *Max-pooling* que toma un *kernel* de 2×2 con un paso de 2 y lo aplica a todo el espacio dimensional de la entrada.

Por último, tenemos a las capas densas, que básicamente funcionan como las capas ocultas de una RNA común, conectando todas las neuronas de la capa densa con todas las neuronas de la capa anterior y de la capa posterior a sí misma.

3.3 YOLO: You Only Look Once.

La operación de las RNC nos permite procesar imágenes de una manera muy eficiente, y gracias a esta tecnología se han desarrollado diversos algoritmos para lograr el reconocimiento de objetos específicos usando este tipo de redes neuronales. Uno de ellos es el llamado YOLO, cuya operación se detalla en el trabajo de (Redmon, Divvala, Girshick, & Farhadi, 2016).

Ese algoritmo divide la imagen de entrada en una malla de *S*×*S*, si el centro del objeto buscado coincide con una de las celdas de la malla, esta celda será la responsable de detectarlo. Cada celda predice un numero de *B* recuadros y un puntaje de confianza para dichos recuadros, con el fin de mostrar con qué nivel de fiabilidad el recuadro en cuestión contiene al objeto que se planea identificar. Si no hay un objeto en la celda, el puntaje de confianza debe ser de cero.

Cada recuadro consiste en cinco predicciones: x, y, w, h y la confianza. El par (x, y) representa a la coordenada del centro del recuadro relativo a sus bordes. el par (h, w) son la altura y la anchura de la celda relativos a la imagen completa. Por último, la confianza de predicción nos indica a la intersección sobre unión (IOU por sus siglas en inglés) entre el recuadro calculado y cualquier recuadro del fondo real.

Además, cada celda calcula una cantidad *C* de probabilidades de clase condicional denotadas por $Pr(Class_i|Object)$. Estas probabilidades están condicionadas en la celda que esté conteniendo un objeto. Solo se calcula un grupo de probabilidades de clase por celda, sin importar la cantidad de recuadros *B*. Para generar los puntajes de confianza específicos para

cada recuadro, se multiplican las probabilidades de clase condicional con las predicciones de recuadro individual como muestra la ecuación (3.34.

$$Pr(Class_i|Object) * Pr(Object) * IOU_{pred}^{truth} = Pr(Class_i)IOU_{pred}^{truth} \quad (3.34)$$

El modelo de YOLO se implementa como una RNC. Las primeras capas convolucionales de la red extraen características de la entrada, mientras que las capas densas calculas las probabilidades y las coordenadas de las salidas. La red de YOLO tiene 24 capas convolucionales seguidas de dos capas densas. Su arquitectura puede verse en la Figura 3.14.

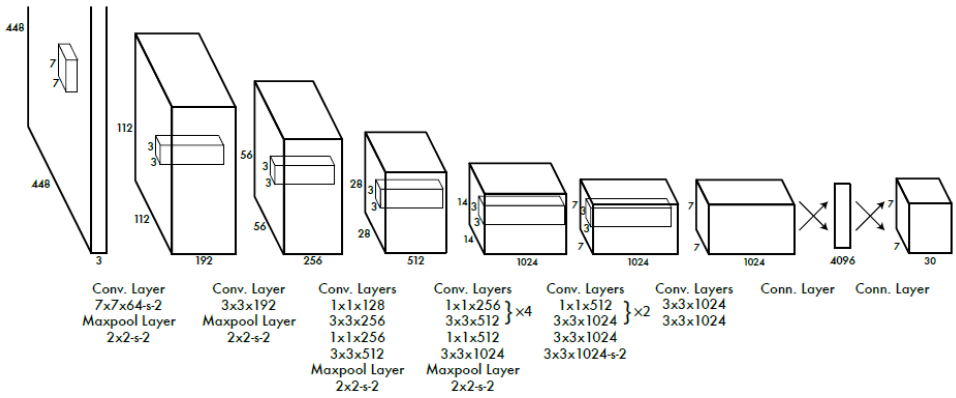


Figura 3.14 Estructura de la red neuronal implementada en YOLO.
Obtenido en: (Redmon, Divvala, Girshick, & Farhadi, 2016)

Se puede apreciar después de toda esta información que el proceso para la identificación automatizada de objetos en imágenes se puede resumir en tres bloques: la obtención información que desmarque al objeto del fondo de la imagen y de otros objetos sin interés para la problemática, la adecuación de esta información de la escena ya procesada, y la construcción del modelo de inferencia por el cual se podrá aprender como diferenciar entre diferentes instancias de un mismo objeto en el entorno. Estos tres bloques son conformados en este trabajo por el procesado de imágenes a partir de operaciones morfológicas, la operación de transformada de Hough y la alimentación de dichas imágenes procesadas a una red neuronal convolucional.

En el siguiente capítulo, se repasarán los entornos de prueba, el sistema desarrollado y la estructura y operación del sistema construido para resolver la problemática planteada en el capítulo I

CAPÍTULO IV. DESARROLLO.

4.1 Condiciones iniciales.

Al momento de iniciar el desarrollo del proyecto, se planteó la opción de probar los sistemas con imágenes tomadas en un entorno de producción real, ya sea buscando algún *dataset* ya armado con imágenes, o tomando las propias dentro de una nave industrial operacional.

Ambas opciones presentaron dificultades. Por un lado, no se encontró un *dataset* que cumpliera las necesidades del proyecto, ya que se tenían muy pocas imágenes de *stock*, mientras que la alternativa de tomar imágenes de una nave industrial fue imposible debido a las limitaciones de entrada a las empresas causadas tanto por la pandemia de Covid-19 vivida al inicio de este trabajo, así como por la falta de apoyo por parte de las empresas, ya que la distribución y funcionamiento de los procesos de producción en una empresa manufacturera estas estrictamente protegidos, por lo que no se permite la toma de imágenes más que para uso interno de dichas empresas.

Tomando estas limitaciones de decidió continuar con el desarrollo en un ambiente de pruebas simulado, como de describe en los siguientes párrafos. Para lograr el funcionamiento del sistema, se definió como el proceso presentado en la Figura 4.1.

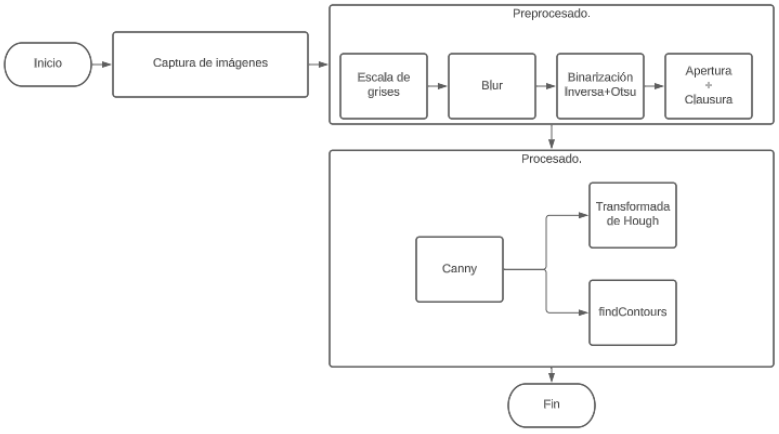


Figura 4.1 Diagrama general del sistema de visión.
Obtenido en: Elaboración propia.

El proceso presentado es el siguiente:

1. Adquirir las imágenes del entorno por medio de capturas con una cámara física.
2. Preprocesado de imágenes para obtener una imagen limpia de los objetos presentes en la imagen.
3. Procesado de imágenes por varias técnicas con el fin de abstraer el contorno de las líneas del pasillo.

Estos pasos son desarrollados a detalle en las secciones subsecuentes de este capítulo.

4.2 Imágenes de prueba.

Como primer paso para definir el ambiente de pruebas, se realizaron varias capturas de imágenes de dos líneas de cinta azul sobre un fondo blanco, capturadas desde un punto fijo con un teléfono móvil con resolución de 640 x 480 pixeles, simplificando el entorno de pasillo en un corredor industrial como se puede ver en la Figura 4.2.

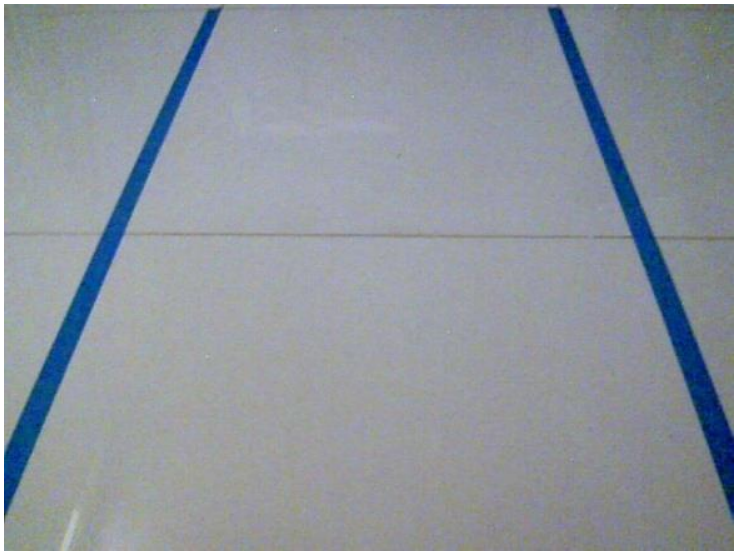


Figura 4.2 Primera versión de imagen de prueba.
Obtenido en: Elaboración propia.

Con el fin de comprobar el comportamiento de los algoritmos desarrollados, se cambió por imágenes como la que se muestra en la Figura 4.3, de 640 x 360 pixeles y tomadas en un entorno a escala más acercado a la realidad y permitiendo diferentes combinaciones de líneas.

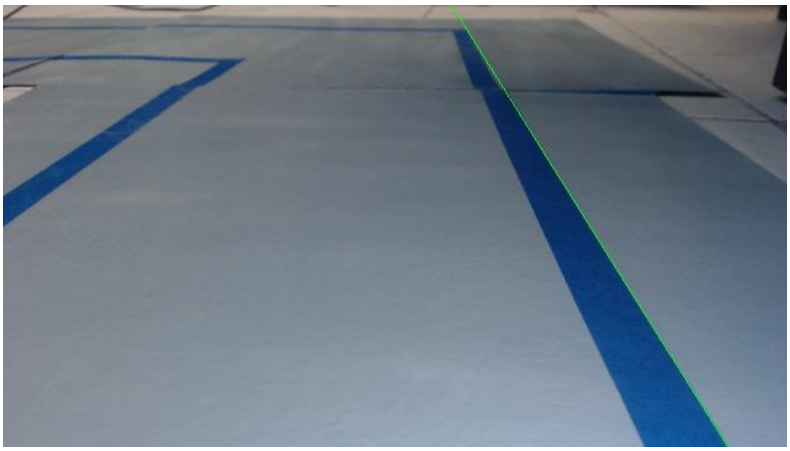


Figura 4.3 Segunda versión de imagen de prueba.
Obtenido en: Elaboración propia.

Estás imágenes fueron capturadas desde una webcam Logitech c920 conectada a una laptop, corriendo los programas en Python 3.8. Así mismo, las imágenes son tomadas desde un robot educacional marca *RoboRobo*, que será descrito con más detalle en una sección posterior a estas, mientras dicho robot está en movimiento. En la Figura 4.4 se muestra dicho robot y el entorno de pruebas.

Figura 4.4 Entorno de captura de imágenes de prueba.
Obtenido en: Elaboración propia.

4.3 Preprocesado de imágenes.

El primer paso es convertir la imagen a escala de grises, en este caso utilizando el método BGR2GRAY de *OpenCV*, seguido por una difuminación y binarización inversa en conjunto con una binarización Otsu. Después se aplica a la imagen una operación de apertura para eliminar ruido en el fondo de la escena seguida de una operación de clausura. El resultado de estas dos operaciones consecutivas es que los objetos presentes con un grosor considerable sean realzados mientras que los objetos más delgados al fondo de la imagen son eliminados. En la Figura 4.5 se muestran los resultados de estas operaciones.

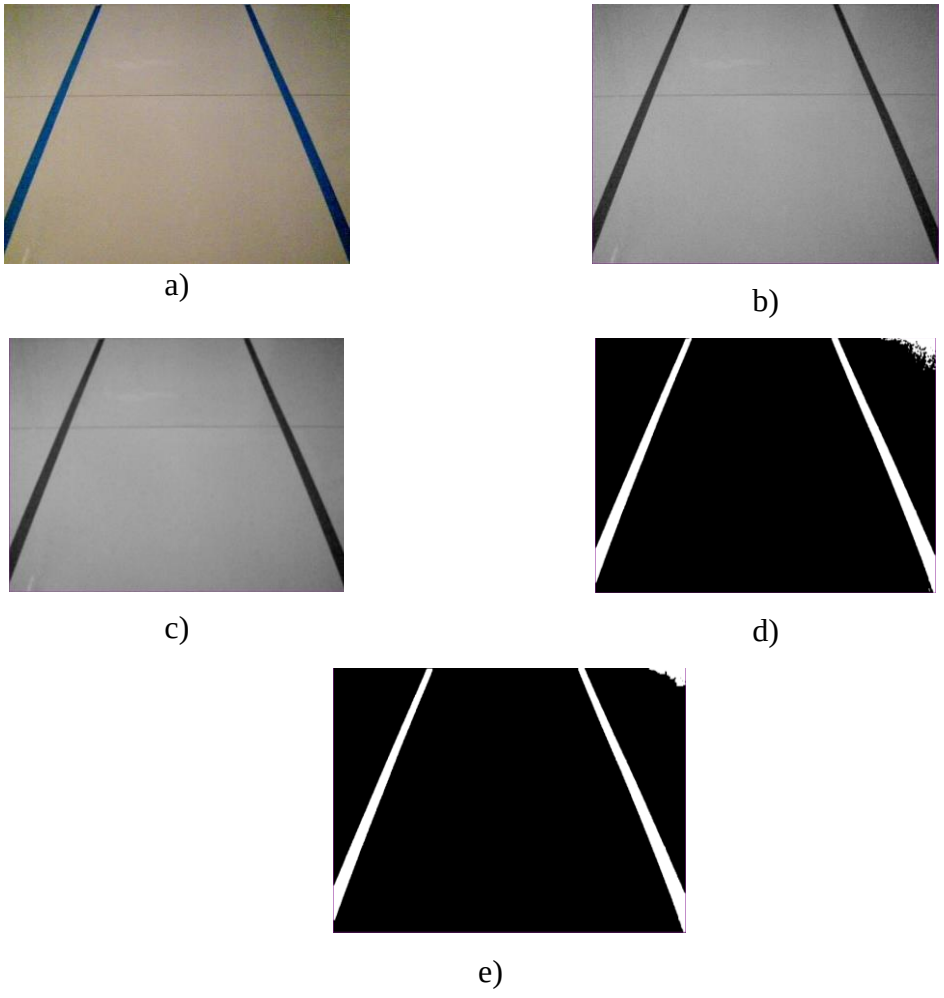
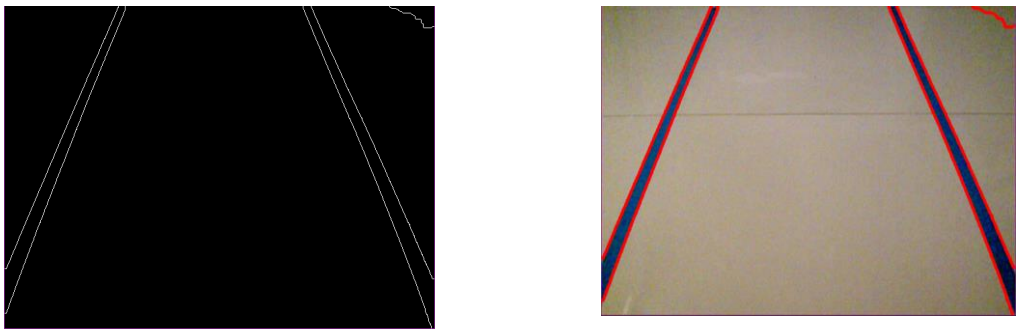


Figura 4.5 Resultados del preprocesado de imagen. a) Imagen base, b) Escala de grises, c) Difuminado, d) Binarización inversa + Otsu y e) Apertura + Clausura.
Obtenido en: Elaboración propia.

4.4 Procesado de imágenes.

Una vez obtenida la imagen con los objetos definidos, se procede a aplicar un filtro de detección de bordes, en este trabajo se utiliza primero la función *Canny* para remarcar los objetos seguida de la función *findContours* de *OpenCV*, teniendo como resultado una imagen negra con los bordes de todos los objetos presentes en la imagen dibujados en blanco. La función *findContours* nos devuelve una lista de vectores donde se guardan los puntos continuos correspondientes a los objetos presentes en la imagen. En la Figura 4.6 se muestra el resultado.



a) b)

Figura 4.6 Resultados de operaciones: a) *Canny* y b) *findContours*.
Obtenido en: Elaboración propia.

4.4.1 Comparación *findContours* y *Houghlines*.

Aquí se presenta una problemática, y es que la función *findContours* también remarca los bordes de objetos tales como sombras sobre el fondo, lo que limita su operación. La manera de funcionar en este caso es aplicar otras funciones de detección más específicas. Para ello, entran en juego las líneas de Hough.

Al utilizar la función *Houghlines* ocurre un comportamiento inestable, debido a que la función genera valores ρ y θ para líneas con muy poca variación entre ellas, lo que produce una imagen con muchas líneas superpuestas. Un ejemplo de esto es la Figura 4.7.

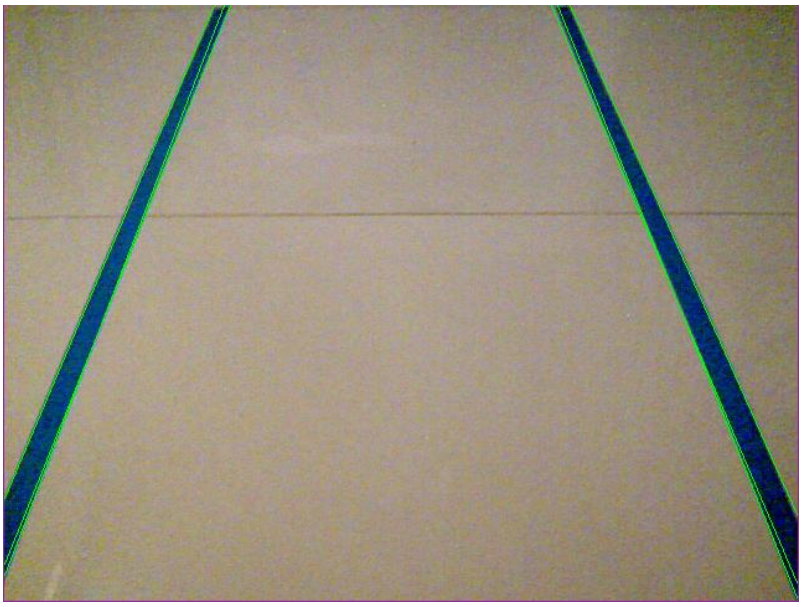


Figura 4.7 líneas detectadas por *HoughLines*.
Obtenido en: Elaboración propia.

4.4.2 Comparación *Houghlines* con promediado y *HoughlinesP*.

OpenCV implementa varias funciones que derivan de la transformada de Hough una de las cuales es *HoughlinesP*, que a diferencia de *Houghlines* entrega los puntos finales de las líneas detectadas en el entorno. Sin embargo, al probarlo dentro del entorno de pruebas toma un comportamiento inestable, representando líneas en partes de la imagen donde no se encuentran ninguna línea. En la Figura 4.8 se aprecian dos cuadros tomados de la misma prueba de funcionamiento, todos tomados con una separación no mayor a 1 segundo.

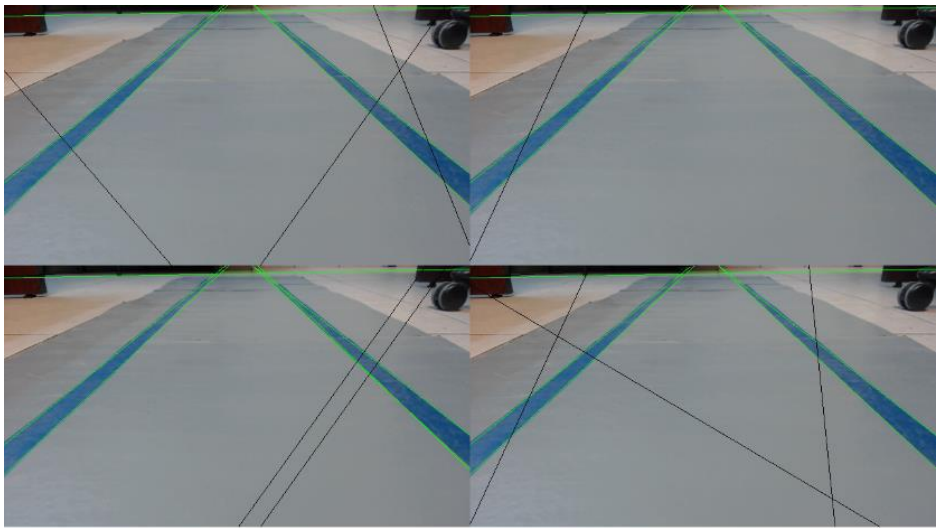


Figura 4.8 Comportamiento de *HoughlinesP*.
Obtenido en: Elaboración propia.

4.4.3 Promediado de líneas.

Para solucionar el problema presentado en la sección 4.4.1 se desarrolló un algoritmo de promediado que separa la lista de valores calculados por *Houghlines* entre valores a la derecha o a la izquierda de la imagen. Esto último se realiza buscando una diferencia de ± 15 unidades entre los valores de θ, poniendo en una lista los valores muy cercanos entre ellos y en otra los demás valores detectados. Este algoritmo puede ser encontrado en el Anexo A.

Teniendo las líneas ya clasificadas por su proximidad se calcula un promedio de sus valores ρ y θ, dando como resultado una única línea por lado de la imagen graficada en color blanco, como se muestra en la Figura 4.9.

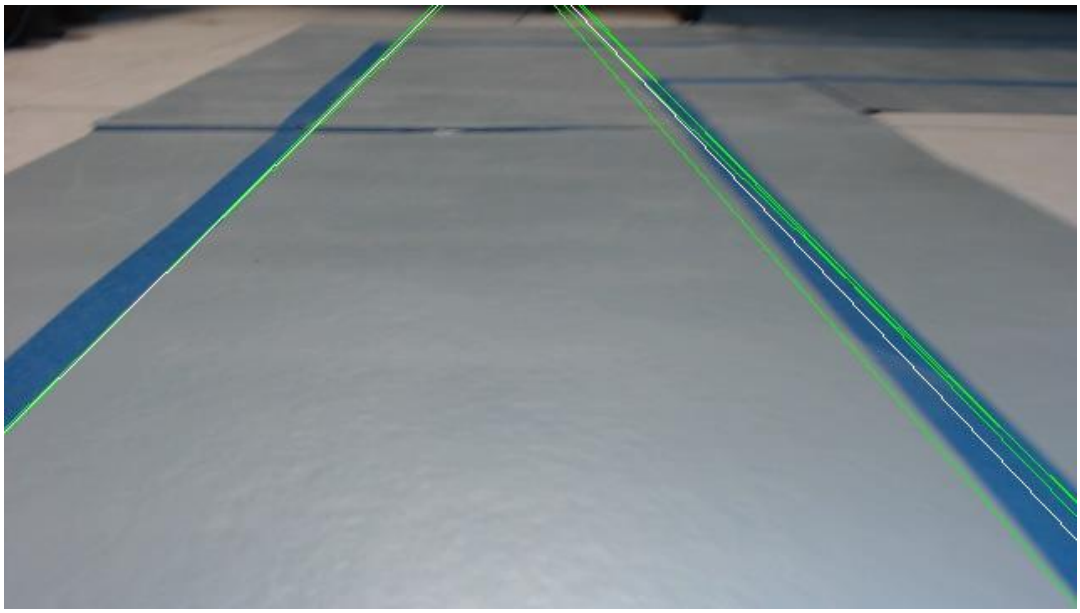


Figura 4.9 Representación de líneas promediadas.
Obtenido en: Elaboración propia.

4.4.4 Abstracción de líneas.

Como último paso del procesado, para abstraer la información relevante del fondo se crea una nueva imagen de fondo negro y las líneas obtenidas graficadas en color blanco, como se muestra en la Figura 4.10.

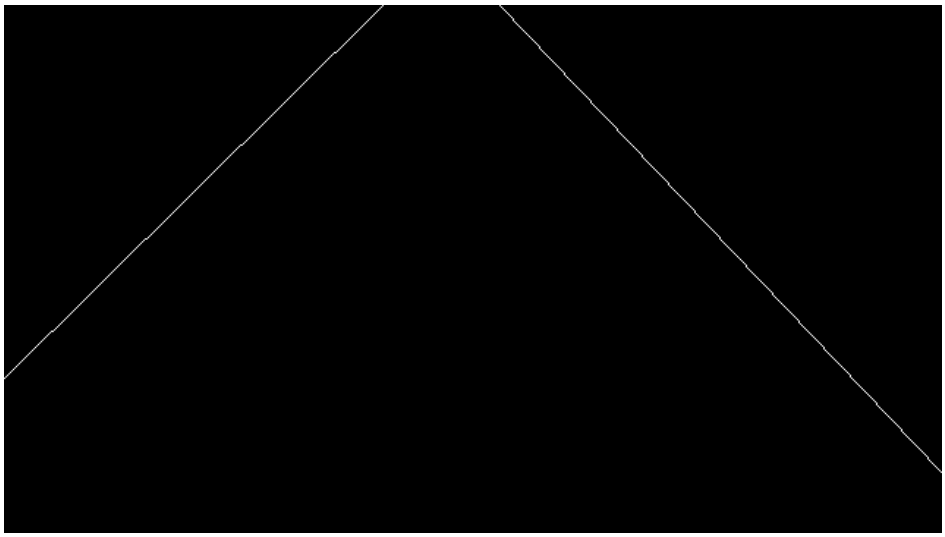


Figura 4.10 Abstracción de las líneas reales obtenida por el algoritmo.
Obtenido en: Elaboración propia.

En ciertos casos, las líneas abstraídas por la transformada de Hough pueden tener la suficiente información para que el sistema pueda tomar decisiones de operación como se ilustra en la Figura 4.10. Sin embargo, hay casos como el que se exhibe en la Figura 4.11 se presenta una ambigüedad difícil de resolver con métodos de computación tradicional.

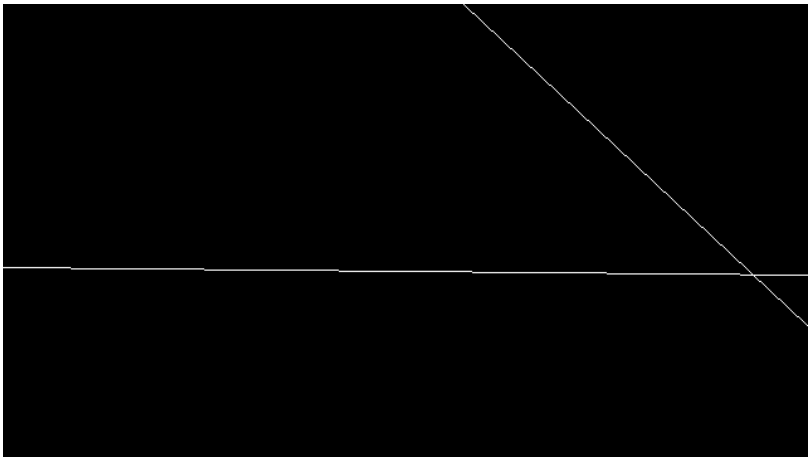
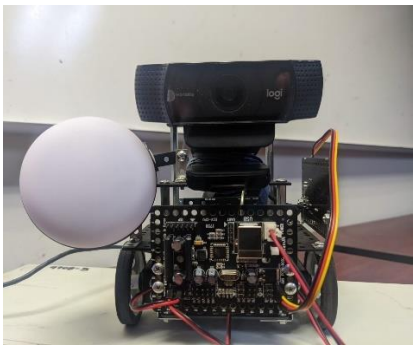


Figura 4.11 Caso de ambigüedad detectado en el proceso de abstracción.
Obtenido en: Elaboración propia.

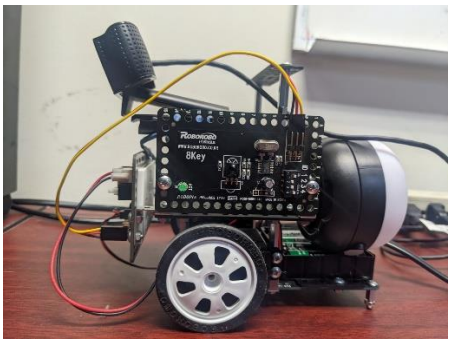
4.5 Entorno simulado para obtención de imágenes de entrenamiento.

Con el propósito de realizar pruebas de funcionamiento del sistema, se construyeron placas de cartón corrugado pintadas de color gris brillante para emular de una manera más fidedigna el entorno de una planta de manufactura industrial, marcando el pasillo con cinta para para enmascarar color azul de una pulgada de ancho. Estas placas son fácilmente ajustables permiten gran variedad de combinaciones para el entorno de pruebas.

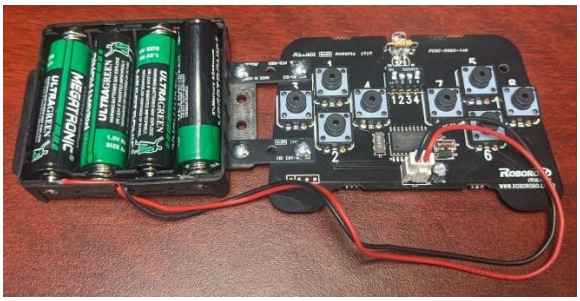
Así mismo, para obtener el punto de vista de un vehículo en movimiento sobre el pasillo se construyó un modelo de vehículo educativo de la marca *RoboRobo*, contando con dos ruedas, una base para colocar la cámara mencionada previamente en este trabajo y una unidad de radiofrecuencia para poder controlar el vehículo a distancia. En primera instancia se equipó al robot con una lámpara portátil para mejorar la iluminación al capturar las imágenes de prueba, pero se eliminó al observar que no hubo un cambio significativo en las capturas a comparación de las capturadas en ausencia de dicha luz. En la Figura 4.12 se muestra dicho robot.



a)



b)



c)

Figura 4.12 Robot educacional para pruebas. a) Vista frontal, b) Vista lateral y c) Control RF.
Obtenido en: Elaboración propia.

Se prepararon varias configuraciones para probar el funcionamiento de los algoritmos aquí descritos. En la Figura 4.13 se muestra uno de los utilizados, ya con el robot en posición para comenzar la prueba.

Figura 4.13 Configuración de ambiente de pruebas.
Obtenido en: Elaboración propia.

4.6 Resultados del procesado.

Una vez se tuvo listo el entorno de pruebas y los programas desarrollados, se pasó a tomar capturas de imágenes y obtener su abstracción de las líneas del pasillo. Para esto, se puso al robot a tomar capturas dentro del entorno mientras se encontraba en movimiento, como se muestra en la Figura 4.14.

Figura 4.14 Ambiente de pruebas configurado para captura de imágenes.
Obtenido en: Elaboración propia.

Se realizaron un total de tres pruebas de captura guardando 3,640 pares de imágenes, correspondientes a la imagen real y a su abstracción. Con el fin de sacar estadísticas de funcionamiento, de manera aleatoria se seleccionaron 344 pares, lo que es equivalente de manera aproximada al 10% de las imágenes capturadas. y estos fueron revisados manualmente para verificar si la abstracción correspondía con su contraparte real.

Para lograr esta validación se tomaron los siguientes puntos como parámetros de evaluación: (Sutherland, 2018)

- Se tienen tres estados en los que se puede valorar una imagen: incorrecto, parcialmente correcto o correcto.
- Para presentar un estado “correcto” las imágenes abstractas deben representar las líneas presentes su par real.

- Se presenta un estado de “parcialmente correcto” si la imagen abstracta no representa a alguna de las líneas presentes en la imagen real, pero si representa a alguna de las demás.
- Se presenta un estado de “incorrecto” al no representar las líneas presentes en la imagen real.

Los resultados obtenidos se muestran en la Tabla 4.1, así mismo, las estadísticas completas se encuentran en el Anexo C.

Tabla 4.1 Resultados generales de pruebas de abstracción.
Obtenido en: Elaboración propia.

Validación	Cantidad	Porcentaje
Incorrecto	88	25.58
Parcialmente correcto	154	44.76
Correcto	102	29.65

Como podemos apreciar, solo con el algoritmo de procesamiento de imágenes, se obtiene menos del 30% de abstracción correcta, lo que presenta una carencia importante en el funcionamiento del algoritmo.

La opción implementada para compensar este funcionamiento, entre las abstracciones parcialmente correctas y los casos de ambigüedad planteados en la sección anterior de este capítulo es simple de plantear: introducir la información obtenida por el procesamiento a una red neuronal para clasificarla en casos concretos y que el sistema pueda devolver una respuesta con la que se pueda tomar una decisión de acción para el caso en particular, lo que deja nuestro sistema como se muestra en la Figura 4.15.

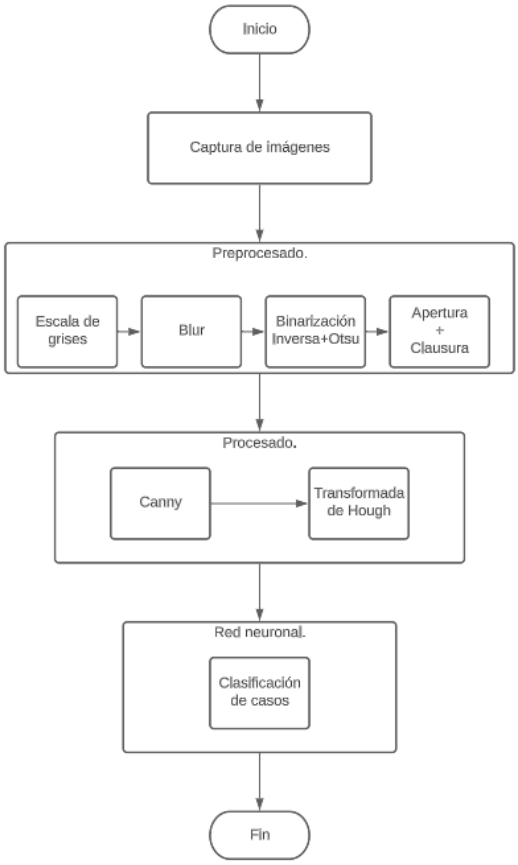


Figura 4.15 Diagrama del sistema de visión con un sistema de clasificación por red neuronal.
Obtenido en: Elaboración propia.

4.7 Etiquetado para entrenamiento de red neuronal.

Como parte del proceso de entrenamiento para una red neuronal de aprendizaje supervisado, se utilizó la herramienta en línea *Roboflow*, con el propósito de definir las clases en las que se pretende que la red neuronal separe los diferentes casos que puedan presentar en una abstracción. Se realizaron dos grupos de entrenamiento, el primero contando las siguientes con cuatro clases:

- end_Path
- move_Forward
- move_Left
- move_Right
- no_Path

Para el segundo *dataset* se removió la clase *no_Path* debido a que el algoritmo de dicha herramienta impide cargar imágenes iguales o demasiado similares entre ellas.

A partir de las imágenes obtenidas del procesado (un total de 3640 imágenes) se cargaron en la herramienta de etiquetado de *Roboflow*. El primer *dataset* a etiquetar quedó reducido a 906 imágenes, mientras que el segundo contiene 2092, como se puede apreciar en la Figura 4.16 y Figura 4.*17*.

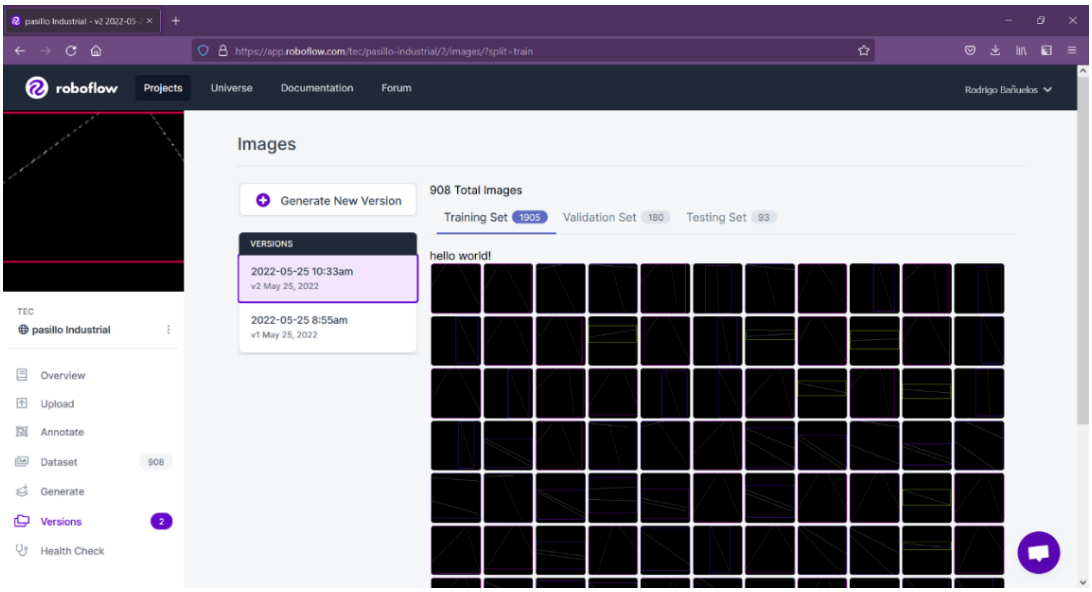


Figura 4.16 Captura de la página de *Roboflow* correspondiente a la primera versión del *dataset*. Obtenido en: Elaboración propia.

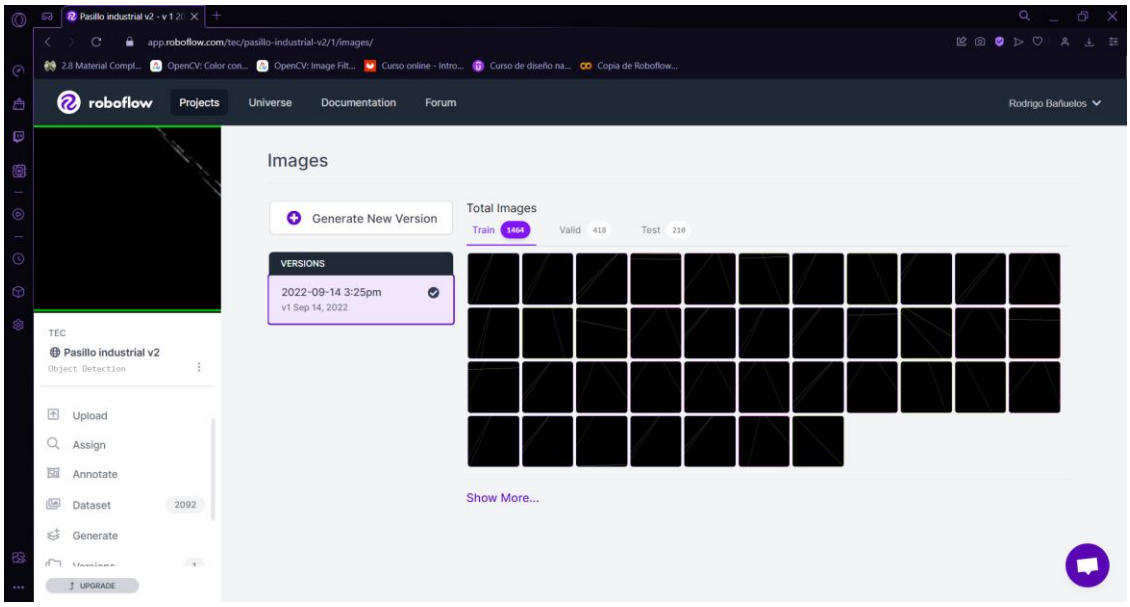


Figura 4.17 Captura de la página de *Roboflow* correspondiente a la segunda versión del *dataset*.
Obtenido en: Elaboración propia.

El etiquetado se realizó de diferente manera para ambos *datsets*. La primera etiquetaba solo las líneas presentes en la imagen como objetos a detectar incluyendo las imágenes sin abstracción presente. En la Figura 4.18 se encuentra un ejemplo de este etiquetado.

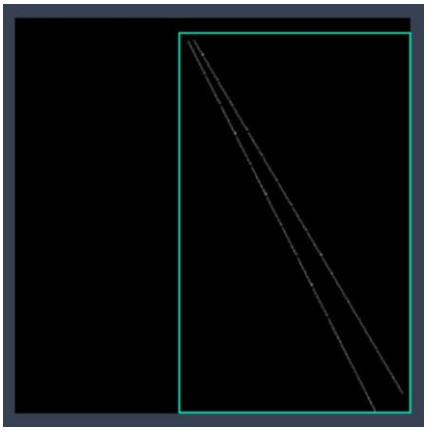


Figura 4.18 Método de etiquetado para la primera versión del *dataset*.
Obtenido en: Elaboración propia.

En la segunda versión del *dataset* no se toma en cuenta la no detección como una clase, sino como una respuesta por defecto. Además, para esta segunda versión se etiqueta toda la imagen como un objeto único en vez de solo las líneas abstraídas para registrar las variaciones en resultados de entrenamiento, como se ve en la Figura 4.19.

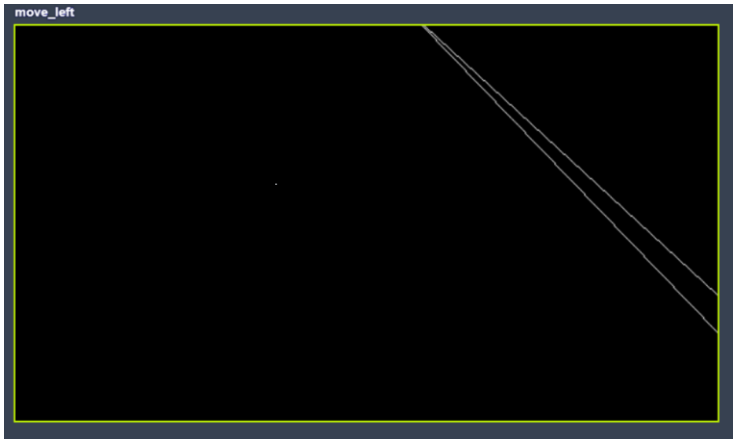


Figura 4.19 Método de etiquetado para la segunda versión del *dataset*.
Obtenido en: Elaboración propia.

4.8 Entrenamiento de red neuronal YOLOv5.

Para empezar el entrenamiento se siguen las instrucciones dadas en la plantilla de *Google Colab*, que es un servicio de compilación en la nube que provee *Google* de manera gratuita o por suscripción, del algoritmo YOLOv5. Se realizaron 40 entrenamientos iniciales con la primera versión del *dataset*, variando los parámetros de *batch* en cantidades iguales a de potencias de 2 y las épocas de entrenamiento como se muestra en la tabla.

Al terminar estos entrenamientos se obtuvieron estadísticas de comportamiento de la red para ajustar aún más el modelo para otros cinco entrenamientos utilizando ahora la segunda versión de *dataset* para verificar si se daba una mejoría en el entrenamiento. Al finalizar dichos entrenamientos se seleccionó aquel con mayor precisión, en este caso fue el entrenamiento numero 2-50, para realizar pruebas de funcionamiento general conectando los sistemas de preprocesado y procesado con el programa de detección de YOLOv5. Estas pruebas se realizaron en el ambiente de pruebas mostrado en este capítulo.

Los resultados obtenidos junto con sus respectivas evidencias son mostrados en el siguiente capítulo.

CAPÍTULO V. RESULTADOS Y DISCUCIÓN.

5.1 Resultados de entrenamiento.

5.1.1 Resultados con el primer *dataset*.

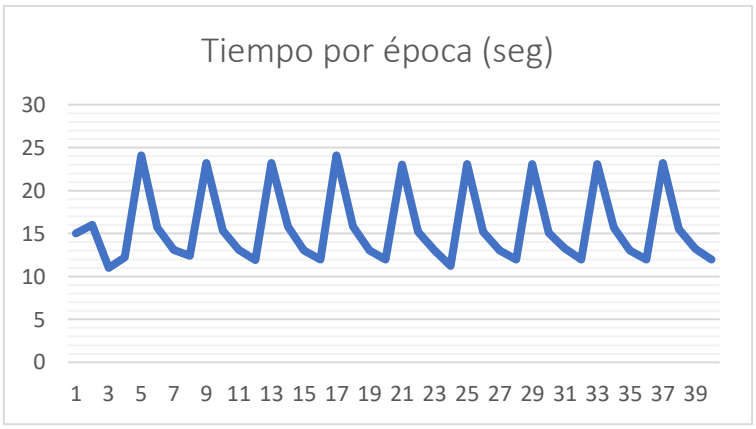
Como se mencionó al final del capítulo anterior, en el proceso de entrenamiento se tomaron los parámetros del modelo al momento de entrenar (tamaño de *batch* y número de épocas) y al finalizar cada entrenamiento se anotó su tiempo de entrenamiento, el tiempo de entrenamiento promedio por época, su precisión máxima alcanzada y la cantidad de imágenes etiquetadas con el segmento de validación del *dataset*. Los resultados registrados se muestran en el Anexo D.

Para apreciar mejor dichos resultados en este trabajo, estos se dividieron en varias gráficas presentadas a continuación.

En la Gráfica 5.1 se puede ver cómo va aumentando el tiempo de entrenamiento conforme aumenta la cantidad de épocas, y también como se aumenta y reduce cíclicamente cada cuatro entrenamientos debido al aumento en el tamaño del *batch* que se utiliza, en este caso, repitiendo el patrón de 8, 16, 32 y 64.

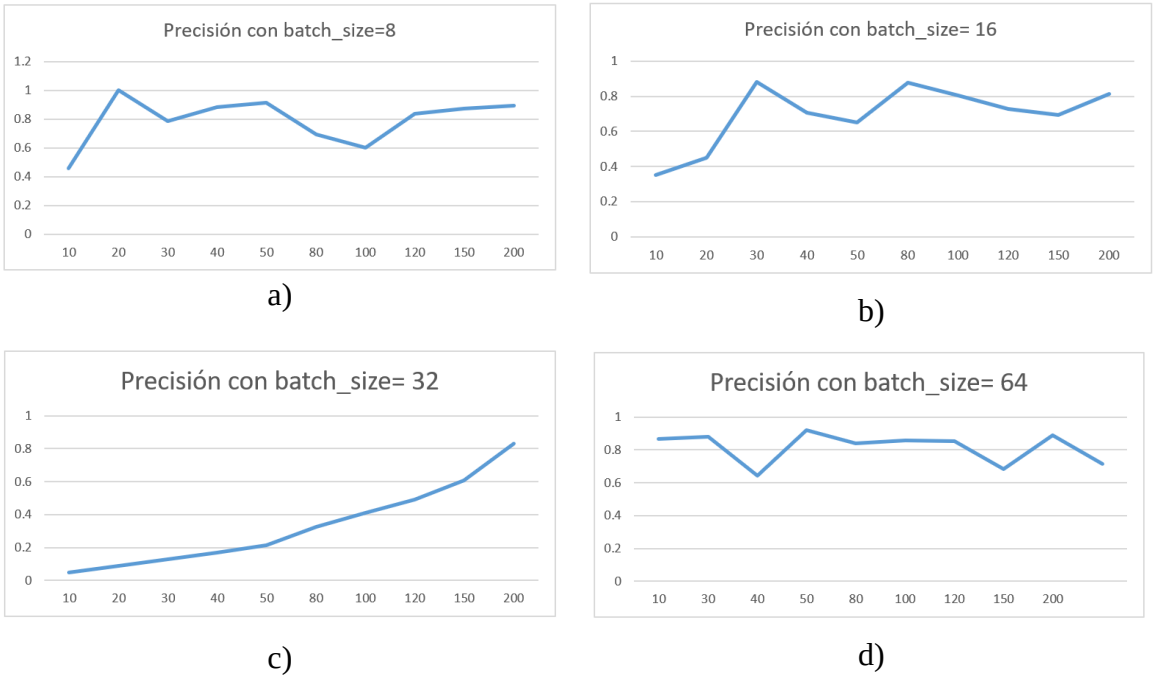
Gráfica 5.1 Tiempo de entrenamiento total por prueba.
Obtenido en: Elaboración propia.

Así mismo, en la Gráfica 5.2 se parecía este mismo ciclo, pero ahora aplicado al tiempo promedio por época.



Gráfica 5.2 Tiempo de entrenamiento promedio por época en cada prueba.
Obtenido en: Elaboración propia.

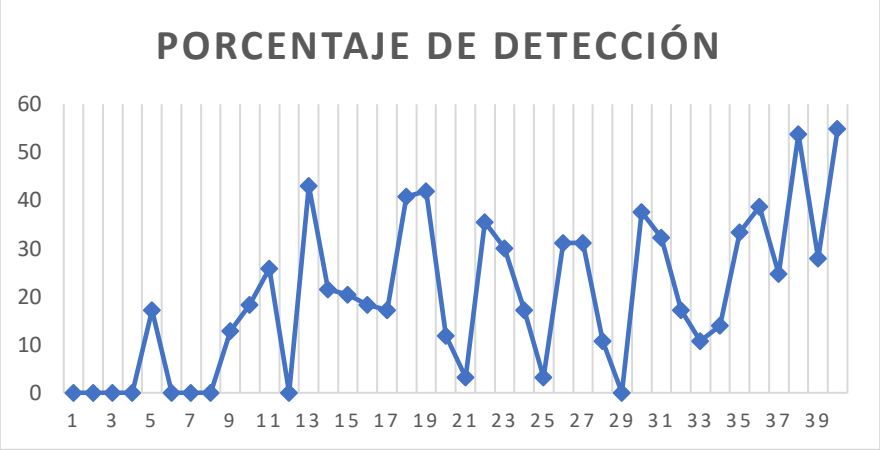
Esto deja claro que entre más grande es el tamaño del *batch* más rápido se completa el entrenamiento, aunque observando la Figura 5.1 la precisión en los entrenamientos con el mimo tamaño de *batch* no se mantiene estable, siendo la mejor por exactitud la de tamaño 64.



a) b) c) d)

Figura 5.1 Gráficas de precisión máxima alcanzada con los diferentes tamaños de batch: a) 8, b) 16, c) 32 y d) 64.
Obtenido en: Elaboración propia.

Con respecto al porcentaje de imágenes etiquetadas con respecto al total de imágenes de prueba, se puede ver sus estadísticas en la Gráfica 5.3.



Gráfica 5.3 Porcentaje de etiquetado de imágenes por prueba.
Obtenido en: Elaboración propia.

Según las pruebas realizadas, el resultado etiquetado no se asocia directamente con el tamaño de batch, sino con el número de épocas, teniendo su mayor número de imágenes etiquetadas en la prueba 40, con 64 de *batch_size* y 200 épocas.

Tomando en cuenta estos resultados, se probaron más combinaciones de tamaños de *batch*, dando los siguientes resultados.

Tabla 5.1 Resultados de entrenamiento con más variaciones de tamaño de *batch*.
Obtenido en: Elaboración propia.

Prueba	Batch	Epochs	Entrenamiento (hr)	Presición	Etiquetado	Porcentaje
41	128	200	0.747	0.8598	77	82.7956989
42	160	200	0.77	0.832	12	12.9032258
43	176	200	0.777	0.7237	40	43.0107527
44	32	500	0.882	~	~	~

Como podemos observar en la Tabla 5.1, las pruebas realizadas con el primer *dataset* confirman que tomar tamaños de *batch* diferentes a potencias de 2 puede mejorar la precisión del modelo con un tope en 192, donde la plantilla de YOLOv5 marca un error de falta de memoria, como se ilustra en la Figura 5.2.

```
%cd /content/yolov5/
!python train.py --img 416 --batch 193 --epochs 250 --data {dataset.location}/data.yaml
```

a)

```
torch.cuda.OutOfMemoryError: CUDA out of memory. Tried to allocate 28.00 MiB (GPU 0; 14.75 GiB total capacity;
CPU times: user 222 ms, sys: 22.3 ms, total: 244 ms
Wall time: 31 s
```

b)

Figura 5.2 a) Línea de código para parametrizar el entrenamiento de la red y b) Error de memoria de la librería Keras.
Obtenido en: Elaboración propia.

5.1.2 Resultados con el segundo *dataset*.

Teniendo estos límites en los parámetros obtenidos por las pruebas anteriores, se realizaron otros cinco entrenamientos con la información ya adquirida de variación de parámetros y el segundo *dataset* obteniendo los siguientes resultados.

Tabla 5.2 Resultados de las pruebas con la segunda versión del *dataset*.
Obtenido en: Elaboración propia.

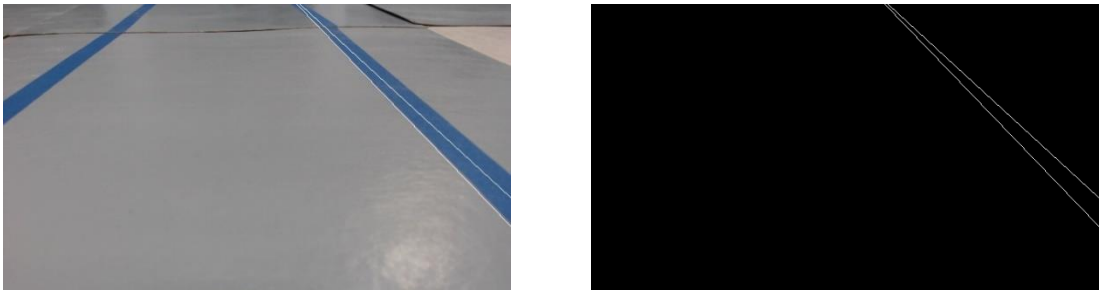
Prueba	Batch	Epochs	Entrenamiento (hr)	Precisión	Etiquetado	Porcentaje
245	128	200	0.433	~	~	~
247	160	300	1.03	0.9268	179	85.2380952
248	160	382	1.391	0.9148	108	51.4285714
249	176	299	1.102	0.9141	170	80.952381
250	176	235	0.967	~	92	43.8095238

En estas pruebas se destaca que, para esta red neuronal, en todos los entrenamientos configurados para ejecutar de 300 épocas en adelante terminan haciendo convergencia antes, entre las 230 y las 290.

Sin embargo, estas últimas pruebas muestran y un aumento significativo en la cantidad de imágenes etiquetadas, alcanzando poco más del 85% de etiquetados.

5.1.3 Descripción de errores significativos.

El sistema presenta algunos casos de error general que se pueden presentar en dos de los bloques principales del trabajo: el procesado de imágenes y el modelo de redes neuronales. Por parte del procesado de imágenes se volvieron muy comunes las capturas donde se abstrae solo una de las líneas, ya sea la de la izquierda o la de la derecha, cuando en la realidad están presentes ambos lados del pasillo marcados, como se puede ver en la Figura 5.3.



a) b)

Figura 5.3 Inconsistencia en la abstracción: a) Imagen real y b) abstracción con representación parcial. Obtenido en: Elaboración propia.

En cuanto a la parte de redes neuronales, se da el caso en que al tomar las abstracciones de imagen real y pasarlas a la red neuronal se etiquetan de manera incorrecta, ya sea que la clase con la que se identifica al objeto presente en la imagen es la clase inadecuada para el objeto o que se presentan dos etiquetas por imagen, una u otra con menos porcentaje de pertenencia a la clase que la otra. Un ejemplo de este caso es el resultado obtenido con el modelo del entrenamiento 249, presentado en la Figura 5.4.

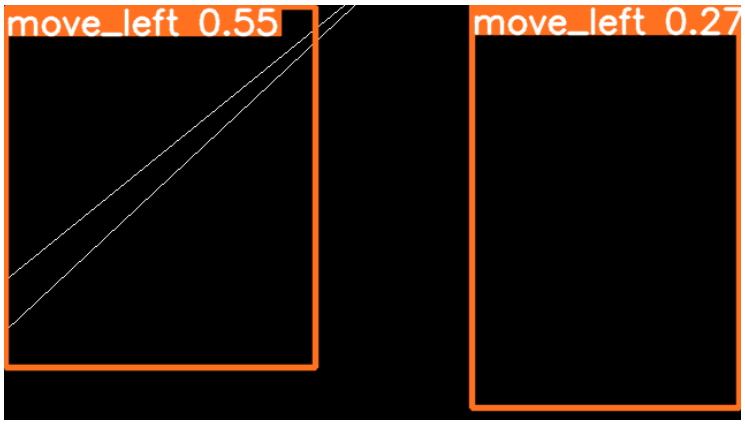


Figura 5.4 Doble etiquetado hecho por el modelo de clasificación.
Obtenido en: Elaboración propia.

CAPÍTULO VI. CONCLUSIONES.

Como se ha repasado en los capítulos I, II y III, las máquinas autónomas presentan una gran oportunidad para volver los procesos industriales más eficientes y seguros, específicamente las que conllevan cargas de elementos de peso considerable. En el caso de este trabajo específico, el tener acceso a un sistema de visión artificial es una parte fundamental para un vehículo autónomo, lo que brinda el primer paso para lograr esta operación independiente del control humano.

Con el desarrollo de este trabajo se dejan claras dos problemáticas asociadas a la construcción y operación de los modelos de inferencia necesarios para el funcionamiento de los vehículos autónomos, las cuales se abordaron de la manera más práctica disponible. Una de estas problemáticas es abstraer la información del entorno relevante para el sistema, en este caso específico, las líneas que marcan el borde del pasillo. Como se remarcó en el capítulo V Resultados y Conclusiones, este proceso de obtención de información relevante se maneja de una manera muy eficiente con la transformada de Hough. Sin embargo, esta técnica tiene el inconveniente de la poca estabilidad al momento ve tomar las capturas del entorno.

La segunda problemática se presenta en el modelo de red neuronal convolucional, debido a que para tener un entrenamiento efectivo que genere un modelo estable es necesario contar con un grupo de datos más variado en cuanto a las posiciones en las que aparecen las líneas del pasillo, así como un incremento en la cantidad de imágenes utilizadas en dicho entrenamiento con el fin de representar fielmente a las clases que se pretenden identificar.

Como alternativas que ayudarían a reducir estas problemáticas se propone aumentar el tamaño del *dataset*, obteniendo imágenes en entornos de prueba más variados en su distribución tanto de líneas como de objetos en los alrededores. También, se plantea la opción de retrabajar el algoritmo de promediado de líneas para ser más preciso al momento de separar las líneas identificadas por la transformada de Hough, de tal manera que representen de mejor manera a las líneas presentes en el entorno.

Aun con lo anterior dicho, este proyecto muestra cómo es posible configurar y probar un sistema de visión artificial con redes neuronales utilizando software y hardware comercial, sin necesidad de implementar equipos de grado industrial, lo cual presenta una gran variedad de posibilidades para trabajos por venir, ya sea para implementar las mejoras previamente mencionadas a este sistema o como una guía para desarrollar un sistema más robusto y completo.

CAPÍTULO VII. BIBLIOGRAFÍA.

Alonso , F. (8 de abril de 2021). *Redes neuronales y Deep Learning. Capítulo 2: La neurona*. Obtenido de Future Space: https://www.futurespace.es/redes-neuronales-y-deep-learning-capitulo-2-la-neurona/

Barker, L., Mills, S., & Langlotz, T. (2016). Power line detection using hough transform and line tracing techniques. *2016 International Conference on Image and Vision Computing New Zealand (IVCNZ)*, 1-6.

Chacón M., M. I. (2007). *Procesamiento digital de imágenes.* Mexico D.F: Editorial Trillas, S.A. de C.V.

Dalitz, C., Schramke, T., & Jeltsch, M. (2017). Iterative Hough transform for line detection in 3D point clouds. *Image processing on line*, 184-196.

García, U. (25 de junio de 2019). *Introducción a las redes neuronales ParteI|Future Lab*. Obtenido de Future Lab: https://futurelab.mx/redes%20neuronales/inteligencia%20artificial/2019/06/25/intro-a-redes-neuronales-pt-1/

González, R., & Woods, R. (1996). *Tratamiento digital de imágenes.* Wilmington, Delawere, E.U.A: Addison-Wesley Iberoamericana, S.A.

Harapanahalli, S., O'Mahony, N., Velasco Hernandez, G., Campbell, S., Riordan, D., & Walsh, J. (2019). Autonomous navigation of mobile robots in factory enviroment. *29th International Conference on Flexible Automation and Intelligent Manufacturing*, 1524-1531.

Hough, P., & Arbor, A. (18 de diciembre de 1962). Methods and Means for recognizing complex patterns. Estados Unidos de America: United States Patent Office.

Jonasson, E. T., Vale, A., & Ramos Pinto, L. (2021). Comparison of three key remote sensing technologies for mobile robot localization in nuclear facilities. *Fusion Engineering and Design*, 1-10.

Larsen, L., Kim, J., Kupke, M., & Schuster, A. (2017). Automatic Path Planning of Industrial Robots Comparing Sampling-Based and Computational Intelligence Methods. *27th International Conference on Flexible Automation and Intelligent Manufacturing*, 241-248.

Li, Q., Adriaansen, A., Udding, J., & Progromsky, A. (2011). Design and control of automated guided vehicle systems: a case study. *Proceedings of the 18th World Congress The International federation of Automatic control*, 13852-13857.

Moreira, A. P., Costa, P., & Lima, J. (2021). New approach for beacons based mobile robot localization using Kalman filters. *30th International Conference on Flexible Automation and Intelligent Manufacturing*, 512-519.

Open Source Computer Vision. (13 de Mayo de 2023). *OpenCV: Color conversions*. Obtenido de OpenCV dopcumentation: https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html

Open source computer Vision. (13 de Mayo de 2023). *OpenCV: image Filtering*. Obtenido de OpenCV Documentation: https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html#ga8c45db9afe636703801b0b2e440fce37

O'shea, K., & Nash, R. (2015). An Introduction to Convolutional Neural Networks. *Arvix Neural and Evolutionary Computing*, 1-11.

Pajares Martinsanz, G., & de la Cruz García, J. (2008). *Visión por computador: imágenes digitales y aplicaciones.* Mexico D.F.: Alfaomega Grupo Editor, S.A de C.V.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time object detection. *CVPR 2016 open access*, 779-788.

Reynolds, A. H. (2019). *Convolutional Neural Networks (CNN's)*. Obtenido de Anh H. Reynolds|Blogs: https://anhreynolds.com/blogs/cnn.html

Sutherland, E. (12 de noviembre de 2018). *Inside Amazon's robot-run warehouse*. Obtenido de Drapers: https://www.drapersonline.com/insight/analysis/inside-amazons-robot-run-warehouse

Zhao, J., Liang, B., & Chen, Q. (2018). The key technologies towards the self-driving car. *International Journal of Intelligent Unmanned Systems*, 2-20.

Zhao, K., Han, Q., Zhang, C.-B., Xu, J., & Cheng, M.-M. (2021). Deep Hough transform for semantic line Detection. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 1-14.

ANEXOS.

Anexo A: Código fuente del sistema.

```
# -*- coding: utf-8 -*-
"""
Created on Thu Nov 18 10:50:14 2021

@author: LRodo
"""

import cv2 as cv
import numpy as np
from time import sleep

cam = cv.VideoCapture(0)
cam.set(3, 480)
cam.set(4, 270)
cam.set(5, 60)

lineas = []
lineasIzq = []
lineasDer = []
meanlines = []

cont = 0

if cam.isOpened():
    width = cam.get(3)
    height = cam.get(4)
    black_img = np.zeros((int(height), int(width), 3), np.uint8)
    black_img2 = np.zeros((int(height), int(width), 3), np.uint8)

def com_Pert(val, array):
    if len(array) == 0:
        return 2
```

```
    else:
        flag = False
        for i in range(len(array)):
            if (val[0] - array[i][0] > -15 and val[0] - array[i][0] < 15) :
                flag = True
            else:
                flag = False
        if flag:
            return 1
        else:
            return 0


def separar_Lineas(lineas, lineasIzq, lineasDer):
    for i in range(0, len(lineas)):
        #print("indice i:" + str(i))
        #print("Valor lineas en i: " + str(lineas[i][0][0]))
        for j in range(0,len(lineas)):
            #print("indice j:" + str(j))
            #print("Valor lineas en j: " + str(lineas[j][0][0]))
            if (lineas[i][0][0] != lineas[j][0][0]):
                #print("+")
                if (lineas[i][0][0]-lineas[j][0][0] > -15 and lineas[i][0][0]-lineas[j][0][0] < 15):
                    #print("-")
                    if (len(lineasIzq) == 0 and len(lineasDer) == 0):
                        lineasIzq.append([lineas[i][0][0], lineas[i][0][1]])
                        #print("Valor " + str(lineas[i][0][0]) + " agregado a LI")
                        lineasIzq.append([lineas[j][0][0], lineas[j][0][1]])
                        #print("Valor " + str(lineas[j][0][0]) + " agregado a LI")
                    else:
                        if not ([lineas[i][0][0], lineas[i][0][1]] in lineasIzq or [lineas[i][0][0],
lineas[i][0][1]] in lineasDer):
                            #print("valor " + str(lineas[i][0][0]) + " duplicado")
                            if com_Pert([lineas[i][0][0], lineas[i][0][1]], lineasIzq) == 1 or
com_Pert([lineas[i][0][0], lineas[i][0][1]], lineasIzq) == 2:
                                lineasIzq.append([lineas[i][0][0], lineas[i][0][1]])
                                #print("Valor " + str(lineas[i][0][0]) + " agregado a LI")
```

```
                            elif com_Pert([lineas[i][0][0], lineas[i][0][1]], lineasDer) == 1 or
com_Pert([lineas[i][0][0], lineas[i][0][1]], lineasDer) == 2:

                                lineasDer.append([lineas[i][0][0], lineas[i][0][1]])

                                #print("Valor " + str(lineas[i][0][0]) + " agregado a LD")

                            else:

                                print("valor " + str(lineas[i][0][0]) + " no compatible")


                        if not ([lineas[j][0][0], lineas[j][0][1]] in lineasIzq or [lineas[j][0][0],
lineas[j][0][1]] in lineasDer):

                            if com_Pert([lineas[j][0][0], lineas[j][0][1]], lineasIzq) == 1 or
com_Pert([lineas[j][0][0], lineas[j][0][1]], lineasIzq) == 2:

                                lineasIzq.append([lineas[j][0][0], lineas[j][0][1]])

                                #print("Valor " + str(lineas[j][0][0]) + " agregado a LI")

                            elif com_Pert([lineas[j][0][0], lineas[j][0][1]], lineasDer) == 1 or
com_Pert([lineas[j][0][0], lineas[j][0][1]], lineasDer) == 2:

                                lineasDer.append([lineas[j][0][0], lineas[j][0][1]])

                                #print("Valor " + str(lineas[j][0][0]) + " agregado a LD")

                            else:

                                print("valor " + str(lineas[i][0][0]) + " no compatible")


                else:

                    if (len(lineasIzq) == 0 and len(lineasDer) == 0):

                        lineasIzq.append([lineas[i][0][0], lineas[i][0][1]])

                        #print("Valor " + str(lineas[i][0][0]) + " agregado a LI")

                        lineasDer.append([lineas[j][0][0], lineas[j][0][1]])

                        #print("Valor " + str(lineas[j][0][0]) + " agregado a LD")

                    else:

                        if not ([lineas[i][0][0], lineas[i][0][1]] in lineasIzq or [lineas[i][0][0],
lineas[i][0][1]] in lineasDer):

                            if com_Pert([lineas[i][0][0], lineas[i][0][1]], lineasIzq) == 1 or
com_Pert([lineas[i][0][0], lineas[i][0][1]], lineasIzq) == 2:

                                lineasIzq.append([lineas[i][0][0], lineas[i][0][1]])

                                #print("Valor " + str(lineas[i][0][0]) + " agregado a LI")

                            elif com_Pert([lineas[i][0][0], lineas[i][0][1]], lineasDer) == 1 or
com_Pert([lineas[i][0][0], lineas[i][0][1]], lineasDer) == 2:

                                lineasDer.append([lineas[i][0][0], lineas[i][0][1]])

                                #print("Valor " + str(lineas[i][0][0]) + " agregado a LD")

                            else:

                                print("valor " + str(lineas[i][0][0]) + " no compatible")
```

```
                        if not ([lineas[j][0][0], lineas[j][0][1]] in lineasIzq or [lineas[j][0][0],
lineas[j][0][1]] in lineasDer):
                            if com_Pert([lineas[j][0][0], lineas[j][0][1]], lineasIzq) == 1 or
com_Pert([lineas[j][0][0], lineas[j][0][1]], lineasIzq) == 2:
                                lineasIzq.append([lineas[j][0][0], lineas[j][0][1]])
                                #print("Valor " + str(lineas[j][0][0]) + " agregado a LI")
                            elif com_Pert([lineas[j][0][0], lineas[j][0][1]], lineasDer) == 1 or
com_Pert([lineas[j][0][0], lineas[j][0][1]], lineasDer) == 2:
                                lineasDer.append([lineas[j][0][0], lineas[j][0][1]])
                                #print("Valor " + str(lineas[j][0][0]) + " agregado a LD")
                            else:
                                print("valor " + str(lineas[i][0][0]) + " no compatible")


def promediar(lineasIzq, lineasDer, meanlines):
    rhoIzq = 0.0
    rhoDer = 0.0
    thetaIzq = 0.0
    thetaDer = 0.0

    if len(lineasIzq):
        for i in range(0, len(lineasIzq)):
            rhoIzq += lineasIzq[i][0]
            thetaIzq += lineasIzq[i][1]

        meanlines.append([rhoIzq/len(lineasIzq), thetaIzq/len(lineasIzq)])
    if len(lineasDer):
        for i in range(0, len(lineasDer)):
            rhoDer += lineasDer[i][0]
            thetaDer += lineasDer[i][1]

        meanlines.append([rhoDer/len(lineasDer), thetaDer/len(lineasDer)])


while(True):
    cont += 1
    ret, frame = cam.read()
```

```
#Preprocesado por Canny
gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)

blur = cv.GaussianBlur(gray,(5,5),0)
#cv.imshow('blur', blur)
ret,th2 = cv.threshold(blur,250,255,cv.THRESH_BINARY_INV+cv.THRESH_OTSU)
#cv.imshow('thresholding', th2)

kernel = np.ones((5,5), np.uint8)
openedimg = cv.morphologyEx(th2, cv.MORPH_OPEN, kernel)
#cv.imshow('Erosion', openedimg)
closedimg = cv.morphologyEx(openedimg, cv.MORPH_CLOSE, kernel)
#cv.imshow('Clausura', closedimg)
edges = cv.Canny(closedimg, 100, 200)
cv.imshow("orillas", edges)

ret, frame3  = cam.read()

lines = cv.HoughLines(edges, 1, np.pi/180, 100)

if (lines is not None):
    for i in range(0, len(lines)):
        rho = lines[i][0][0]
        theta = lines[i][0][1]
        a = np.cos(theta)
        b = np.sin(theta)
        x0 = a * rho
        y0 = b * rho
        pt1 = (int(x0 + 1000*(-b)), int(y0 + 1000*(a)))
        pt2 = (int(x0 - 1000*(-b)), int(y0 - 1000*(a)))
        cv.line(frame3, pt1, pt2, (0,255,0), 1)

    for i in range(0, len(lines)):
        aux= lines[i].tolist()
        lineas.append(aux)
```

```
        separar_Lineas(lines, lineasIzq, lineasDer)


        print("lineas Izquierda:")
        for i in range(0, len(lineasIzq)):
            print(str(lineasIzq[i]))


        print("lineas Derecha:")
        for i in range(0, len(lineasDer)):
            print(str(lineasDer[i]))


        promediar(lineasIzq, lineasDer, meanlines)


        print(meanlines)


        for i in range(0, len(meanlines)):
            rho = meanlines[i][0]
            theta = meanlines[i][1]
            a = np.cos(theta)
            b = np.sin(theta)
            x0 = a * rho
            y0 = b * rho
            pt1 = (int(x0 + 1000*(-b)), int(y0 + 1000*(a)))
            pt2 = (int(x0 - 1000*(-b)), int(y0 - 1000*(a)))
            cv.line(frame3, pt1, pt2, (255,255,255), 1)
            cv.line(black_img, pt1, pt2, (255,255,255), 1)



    cv.imshow("lineas de Hough", frame3)
    cv.imshow("Imagen negra", black_img)
    #cv.imwrite(str(cont) + '.png', black_img)
    #cv.imwrite(str(cont) + '.jpg', frame3)
    black_img = np.zeros((int(height), int(width), 3), np.uint8)
    black_img2 = np.zeros((int(height), int(width), 3), np.uint8)


    if cv.waitKey(1) & 0xFF == ord('q'):
        break
```

```
    print(str(cont))

cam.release()

cv.destroyAllWindows()
```

Anexo B: Enlaces de herramientas utilizadas.

https://colab.research.google.com/github/roboflow-ai/notebooks/blob/main/notebooks/train-yolov5-object-detection-on-custom-data.ipynb

Datasets en *Roboflow*.

Primera versión:

https://app.roboflow.com/tec/pasillo-industrial/2

Segunda versión:

https://app.roboflow.com/tec/pasillo-industrial-v2/deploy/1

Anexo C: Tabla de resultados de la abstracción.

Tabla 1 Resultados para validación “Incorrecta”.

Notas	Cantidad	Porcentaje
Sin detección	82	93.18
Representación incompleta	0	0
Falta línea derecha	0	0
Falta línea izquierda	0	0
Angulo con varianza entre línea real y abstracta	0	0
Sin notas	0	0
Detección de línea no presente	6	6.82
Detección de 2 bordes	0	0
Total de imágenes	88	100

Tabla 2 Resultados para validación “Parcialmente correcta”.

Notas	Cantidad	Porcentaje
Sin detección	0	0
Representación incompleta	23	14.94
Falta línea derecha	78	50.65
Falta línea izquierda	45	29.22
Ángulo con varianza entre línea real y abstracta	2	1.30
Sin notas	0	0
Detección de línea no presente	6	3.90
Detección de 2 bordes	0	0
Total de imágenes	154	100

Tabla 3 Resultados para validación “Correcta”.

Notas	Cantidad	Porcentaje
Sin detección	1	0.98
Representación incompleta	2	1.96
Falta línea derecha	0	0
Falta línea izquierda	0	0
Ángulo con varianza entre línea real y abstracta	30	29.41
Sin notas	54	52.94
Detección de línea no presente	0	0
Detección de 2 bordes	15	14.71
Total de imágenes	102	100

Anexo D: Tabla de resultado de entrenamiento.

Prueba	*Batch*	Épocas	Tiempo de entrenamiento (hr)	Precisión máxima	Etiquetado	Porcentaje
1	16	10	0.052	0.35	0	0
2	8	10	0.074	0.4588	0	0
3	32	10	0.047	0.8147	0	0
4	64	10	0.043	0.8663	0	0
5	8	20	0.146	0.9994	16	17.20430108
6	16	20	0.101	0.4481	0	0
7	32	20	0.088	0.8571	0	0
8	64	20	0.081	0.8799	0	0
9	8	30	0.215	0.7861	12	12.90322581
10	16	30	0.149	0.8832	17	18.27956989
11	32	30	0.129	0.8428	24	25.80645161
12	64	30	0.12	0.6411	0	0
13	8	40	0.283	0.8849	40	43.01075269
14	16	40	0.195	0.7085	20	21.50537634
15	32	40	0.168	0.9196	19	20.43010753
16	64	40	0.168	0.9195	17	18.27956989
17	8	50	0.363	0.9166	16	17.20430108
18	16	50	0.247	0.6517	38	40.86021505
19	32	50	0.212	0.876	39	41.93548387
20	64	50	0.199	0.8381	11	11.82795699
21	8	80	0.557	0.6939	3	3.225806452
22	16	80	0.378	0.8779	33	35.48387097
23	32	80	0.327	0.8328	28	30.10752688

24	64	80	0.303	0.8565	16	17.20430108
25	8	100	0.692	0.6001	3	3.225806452
26	16	100	0.477	0.8041	29	31.1827957
27	32	100	0.412	0.9131	29	31.1827957
28	64	100	0.384	0.8521	10	10.75268817
29	8	120	0.837	0.836	0	0
30	16	120	0.57	0.7263	35	37.6344086
31	32	120	0.492	0.7263	30	32.25806452
32	64	120	0.459	0.6823	16	17.20430108
33	8	150	1.049	0.8755	10	10.75268817
34	16	150	0.731	0.6916	13	13.97849462
35	32	150	0.607	0.9974	31	33.33333333
36	64	150	0.573	0.891	36	38.70967742
37	8	200	1.394	0.8918	23	24.7311828
38	16	200	0.97	0.8138	50	53.76344086
39	32	200	0.829	0.8991	26	27.95698925
40	64	200	0.769	0.7155	51	54.83870968
41	128	200	0.747	0.8598	77	82.79569892
42	160	200	0.77	0.832	12	12.90322581
43	176	200	0.777	0.7237	40	43.01075269
44	32	500	0.882	~	~	~
245	128	200	0.433	~	~	~
247	160	300	1.03	0.9268	179	85.23809524
248	160	382	1.391	0.9148	108	51.42857143
249	176	299	1.102	0.9141	170	80.95238095
250	176	235	0.967	~	92	43.80952381