Universidad Internacional de La Rioja (UNIR)

ESIT

Máster Universitario en Inteligencia Artificial

Detección y clasificación de obstáculos mediante visión estéreo en la toma inteligente de decisiones para robots autónomos en espacios reducidos

Trabajo Fin de Máster

Presentado por: Báez Maldonado, José Daniel

Director: Dr. Salvador Cobos Guzman

Ciudad: Portoviejo
Fecha: 2020-09-22

Resumen

Mientas aumenta el uso de robots de navegación autónoma dentro de ambientes compartidos con humanos, se vuelve más evidente la necesidad de mejorar las medidas de seguridad que éstos integran para que su movilidad no represente un riesgo para personas y animales. Este trabajo presenta una herramienta de software para la detección y clasificación de obstáculos cercanos mediante algoritmos de visión artificial, orientado a la toma de decisiones que protejan la integridad física de niños y mascotas que se encuentren en el entorno de robots autónomos. Se utilizan mapas de profundidad generados por una cámara estéreo para identificar la ruta crítica y los obstáculos presentes, junto con una red neuronal convolucional para su clasificación. Una etapa final de toma de decisiones recopila la información obtenida para sugerir las acciones más adecuadas de acuerdo a cada caso, explorando la posibilidad de manipular los objetos no-críticos en caso de ser necesario.

Palabras Clave: Aprendizaje profundo, detección, navegación autónoma, obstáculo, visión estéreo.

Abstract

As the use of autonomous navigation robots within human environments increases, the need to improve the security measures they integrate so that their mobility does not represent a risk for people and animals, becomes more evident. This work presents a software tool for the detection and classification of nearby obstacles using artificial vision algorithms, aimed at making decisions that protect the physical integrity of children and pets that are in the around autonomous robots. Depth maps generated by a stereo camera are used to identify the critical path and obstacles, along with a convolutional neural network for their classification. A final decision-making stage collects the information obtained to suggest the most appropriate actions according to each case, exploring the possibility of manipulating non-critical objects if necessary.

Keywords: Deep learning, detection, autonomous navigation, obstacle, stereo vision.

Índice de contenidos

Índice de tablas

Índice de figuras

1. Introducción

La navegación autónoma ha logrado que muchos robots móviles y vehículos no tripulados se desplacen por sí mismos dentro de un entorno conocido o desconocido, para cumplir con sus tareas asignadas. Así mismo, el continuo desarrollo de la visión computarizada de la mano con el aprendizaje automático, ha permitido que las máquinas sean capaces de reconocer algunos de los objetos que están siendo capturados por una o varias cámaras. El conjunto de estos avances ha conseguido dotar de cierto grado de inteligencia a los robots, que haciendo uso de toda esta información pueden decidir con mejor criterio las tareas que se deben ejecutar.

Una de estas tareas es la evasión de obstáculos, cuya importancia ha crecido notablemente y está siendo ampliamente usada en aplicaciones que van desde manipuladores robóticos industriales hasta vehículos autónomos. En el ámbito industrial, la manipulación de obstáculos es un campo de exploración que ha obtenido buenos resultados hasta el momento y continúa en desarrollo. Sin embargo, la posibilidad de manipulación de ciertos obstáculos durante la navegación autónoma de robots móviles presenta una gran oportunidad de innovación, ya que es un tema poco explorado mientras que puede aportar numerosas ventajas en espacios cerrados. Ejemplos de estos entornos son departamentos, casas, oficinas, edificios, y hospitales, donde cada día es más común la colaboración de agentes robóticos en actividades como limpieza y desinfección.

1.1 Motivación

El uso de robots “inteligentes” con navegación autónoma se ha expandido notablemente durante los últimos años, hasta convertirse en herramientas cotidianas de trabajo en distintas aplicaciones que incluyen desde aspiradoras de uso doméstico hasta plataformas transportadoras de material para uso industrial. Mientras aumenta la popularidad de este tipo de robots también se vuelve más evidente la necesidad de mejorar las medidas de seguridad que integran, con el fin de salvaguardar la integridad física de las personas y animales que forman parte de los entornos donde éstos se desplazan.

Una forma de dar solución a esta problemática es la detección de dichos individuos potencialmente vulnerables, y así evitar la ejecución de acciones que puedan suponer un riesgo para éstos. Este reconocimiento es posible con el uso de clasificadores entrenados mediante técnicas de aprendizaje automático y visión computarizada, y puede guiar a tomar las decisiones adecuadas en base a un razonamiento lógico sencillo.

1.2 Planteamiento del trabajo

Este trabajo busca proporcionar un método de detección de personas y mascotas que se encuentren cercanas al robot, con el fin de realizar las acciones adecuadas que no representen un peligro a su seguridad. Para lograr esto, se utilizará la información contenida en el mapa de profundidades generado por visión estéreo, en conjunto con un modelo de clasificación de imágenes utilizando técnicas de aprendizaje profundo.

Adicionalmente, se explora la posibilidad de manipulación de objetos no-críticos en caso de ser necesario, utilizando el modelo inteligente para realizar una clasificación binaria del tipo de obstáculo. La implementación y pruebas se realizarán en una tarjeta *StereoPi*, que permite la visión estereoscópica gracias a su compatibilidad de conexión de dos cámaras con el módulo de cómputo *Raspberry Pi.*

1.3 Estructura de la memoria

En el capítulo 2 se realiza un repaso de los trabajos destacados en el estado del arte referente a la navegación autónoma y evasión de obstáculos. También se mencionan publicaciones que hacen uso de la visión estéreo en sistemas de localización y mapeo simultáneos SLAM. Se parte desde un contexto general hasta los desarrollos más cercanos al presente trabajo mediante el uso de técnicas de inteligencia artificial y cámaras estereoscópicas para el reconocimiento y manipulación de objetos.

Los objetivos y la metodología de trabajo que direccionan este trabajo se detallan en el capítulo 3, mientras que el capítulo 4 identifica los requisitos que satisface la herramienta de software propuesta, cuyos detalles de desarrollo son descritos en el capítulo 5.

Finalmente, en el capítulo 6 se muestran los resultados obtenidos en las pruebas realizadas sobre la implementación del algoritmo desarrollado y en el capítulo 7 se describen las conclusiones a las que se ha llegado respecto al desarrollo y evaluación de la presente propuesta.

2. Contexto y estado del arte

En las últimas décadas, la robótica ha presentado un gran crecimiento; poco a poco, los robots se están convirtiendo en compañeros cotidianos de las personas. La interacción y convivencia humano-máquina se vuelve evidente conforme el continuo desarrollo permite la creación de robots más “inteligentes”. Un gran ejemplo son los robots móviles con navegación autónoma. Una gran variedad de sistemas y características se combinan con el fin de proporcionar dicho grado de inteligencia y autonomía, que les permite tomar decisiones y ejecutar acciones de acuerdo a sus funcionalidades y su entorno. Estos sistemas incluyen desde operaciones sencillas de sensado hasta algoritmos complejos de procesamiento de información.

2.1. Visión estéreo

El término “visión estéreo” se define como el proceso que combina múltiples imágenes de una escena para extraer información geométrica tridimensional. Su versión más común y sencilla utiliza solamente dos imágenes, se conoce también como visión binocular y está directamente inspirada en el sistema visual de los humanos y animales (Ayache, 1991). Gracias a la ubicación de los ojos en el rostro humano, la información captada y enviada al sistema nervioso central son dos imágenes similares del entorno, tomadas desde dos puntos cercanos en el mismo nivel horizontal. La posición de un objeto en una imagen difiere respecto a la otra, dependiendo de la distancia a la que se encuentra del observador, como se observa en la *Figura 1*. El cerebro es capaz de medir esta disparidad y usarla para estimar la profundidad a la que se encuentra cada objeto que se observa (Marr & Poggio, 2013).

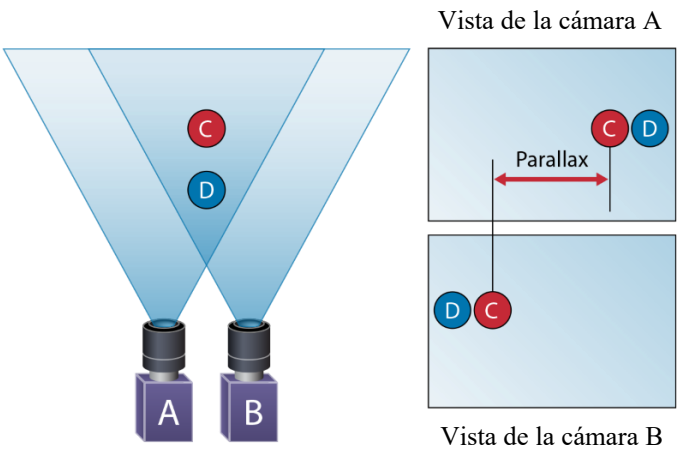


Figura 1. *Diferencias en las vistas de la visión estéreo* (ams AG, 2020).

Los algoritmos de visión estéreo, también conocida como visión estereoscópica, buscan modelar computacionalmente esta función que es ejecutada de manera natural por las personas. Para lograrlo, (Marr & Poggio, 2013) establecen tres pasos principales: se selecciona un punto o región pequeña en una de las imágenes; el mismo punto debe ser identificado en la otra imagen; y finalmente se mide la disparidad entre los puntos correspondientes. Estos pasos se deben realizar para todos los puntos que se consideren necesarios para una buena interpretación del entorno. Dependiendo de los requerimientos de la aplicación, se puede calcular la profundidad a partir de la disparidad encontrada.

2.1.1. Disparidad

La disparidad se define como la diferencia de localización en imagen de un mismo punto tridimensional, cuando es proyectado bajo la perspectiva de dos cámaras distintas (UW CSE vision faculty, 2017). En el bosquejo mostrado en la *Figura 2*, el cuadro resaltado representa un punto tridimensional capturado por una cámara estereoscópica. La localización de dicho punto en el eje vertical es igual en ambas imágenes, pero su localización horizontal varía según la perspectiva.

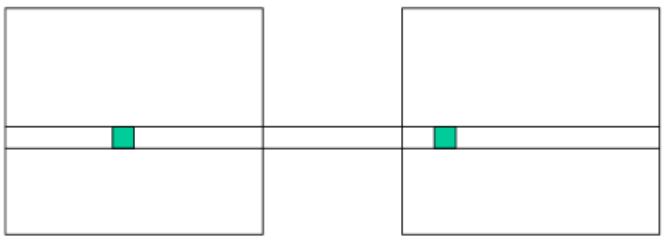


Figura 2. *Disparidad de un punto tridimensional* (UW CSE vision faculty, 2017).

Si la localización horizontal de este punto en la vista izquierda es x_{izq} y en la vista derecha es x_{der}, la disparidad d que éste presenta se obtiene mediante la diferencia de ambos valores, como se indica en la *Ecuación 1*.

(Ecuación 1)

$$d = x_{izq} - x_{der}$$

Sin embargo, conocer la disparidad de un solo punto no es suficiente para tener una buena comprensión de la tridimensionalidad de la escena. Es necesario generar un mapa de disparidad que pueda representar la totalidad, o una porción significativa, del entorno observado.

2.1.2. Mapas de disparidad y profundidad

Encontrar las disparidades adecuadas a partir de dos imágenes no es un proceso computacionalmente sencillo, ya que se puede presentar una gran cantidad de ambigüedades. Un esquema de los pasos involucrados en el desarrollo de un algoritmo para visión estéreo es presentado por (Scharstein, Szeliski, & Zabih, 2001), como se indica en la *Figura 3*.

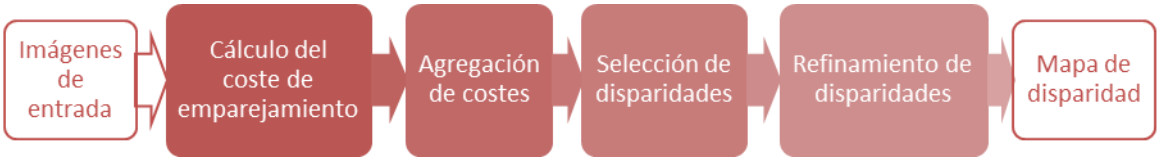


Figura 3. *Pasos en el desarrollo de algoritmos de visión estéreo*.

Estos pasos son realizados se mantienen constantes en dos tipos de algoritmos, clasificados como locales y globales. Los **métodos locales** se caracterizan por requerir un tiempo menor, debido a que el cálculo de la disparidad de un pixel depende únicamente de los valores de intensidad en una ventana de tamaño predefinido. La agregación de costes corresponde a la suma o promedio en dicha ventana. La disparidad que se asigna a cada pixel es la correspondiente al menor coste. Existen algunos algoritmos que, como casos particulares, ejecutan una comparación pixel a pixel, como el propuesto por (Aslam & Ansari, 2019).

Los **métodos globales**, por su parte, tratan la asignación de disparidades como un problema de minimización de una función de energía global para todos los valores de disparidad (Hamzah & Ibrahim, 2016). Los buenos resultados que éstos presentan son compensados por sus altos requerimientos computacionales, que los hacen poco útiles para aplicaciones en tiempo real.

La salida del proceso de la *Figura 3* es un mapa de disparidad, que generalmente es del mismo tamaño que las imágenes de entrada. Este mapa contiene el valor de disparidad calculado para cada uno de los pixeles emparejados. El objetivo es obtener representaciones como las que se muestran en la *Figura 4*, donde el color rojo representa valores altos de disparidad, y el color azul representa valores bajos de disparidad.

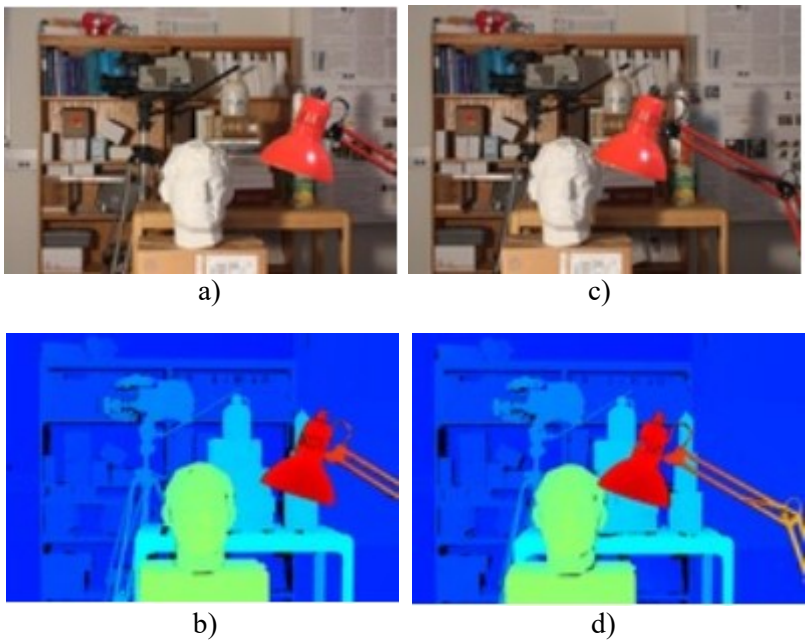


Figura 4. *Newkuba del dataset Middlebury* (Middlebury College, 2015):
a) imagen izquierda, b) mapa de disparidad de la imagen izquierda,
c) imagen derecha, d) mapa de disparidad de la imagen derecha.

Como se puede observar, la profundidad de cada punto es inversamente proporcional a su disparidad correspondiente. De esta manera, mediante un cálculo simple, el mapa de disparidad puede ser transformado en un mapa de profundidades estimadas. Por esta razón, algunas fuentes no hacen distinción entre estos dos términos.

2.1.3. Cámaras estereoscópicas

El sistema visual humano es la herramienta de visión más sofisticada y poderosa para observar el entorno y extraer información. La visión estéreo es el sistema artificial, construido para aplicaciones de robótica, que busca asemejarse a la visión biológica (IntoRobotics, 2013). Esto se consigue a través de dispositivos conocidos como cámaras estereoscópicas, cámaras estéreo, sensores de visión estéreo, o cámaras de profundidad. En similitud a los ojos de una persona, las cámaras estéreo se componen de dos cámaras que observan vistas ligeramente distintas del mismo entorno, debido a la separación que existe entre ellas.

Existe una gran variedad de sensores desarrollados para la visión estéreo. Según la aplicación, se pueden utilizar cámaras de visión amplia, con distancia focal fija o variable, cámaras con distintas resoluciones, velocidades de captura (*FPS*), y compatibilidad con tarjetas electrónicas. Entre estas opciones se encuentran las cámaras de profundidad de la serie *RealSense* (Intel Corporation, 2020), cuyo modelo *D455* se muestra en la *Figura 5*.

Figura 5. *Cámara de profundidad Intel RealSense D455*.

2.1.3.1. StereoPi

La tarjeta de desarrollo *StereoPi* (*Figura 6*) fue creada por el equipo (virt2real, 2020) como una propuesta de bajo costo para proyectos de visión estéreo. Está basada en *Raspberry Pi*, por lo que presenta una compatibilidad casi directa con los sistemas Linux adaptados para *Raspberry Pi* como Raspbian, y algunas versiones de Ubuntu. Dicha compatibilidad es una de las principales características que ha provocado que *StereoPi* gane popularidad entre los desarrolladores e investigadores de aplicaciones con software libre. Adicionalmente, las librerías para *Python* que han sido adaptadas para esta tarjeta, hacen que sea muy fácil empezar a generar código personalizado. De esta manera, se puede hacer uso de los beneficios de la visión estéreo según las necesidades de cada aplicación.

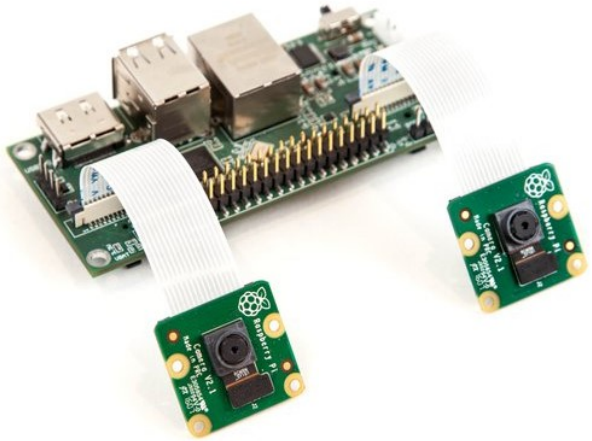


Figura 6. *Tarjeta de desarrollo y cámara estéreo: StereoPi.*

Entre los proyectos que han sido desarrollados utilizando una *StereoPi* está la transmisión en directo de videos 3D a través de YouTube, construcción de mapas de profundidad usando *OpenCV*, mapas de profundidad usando *ROS (Robot Operating System)*, fotos y videos panorámicos de 360°, transmisión tridimensional desde drones, aplicaciones de realidad virtual, realidad aumentada, entre otros.

2.2. Reconocimiento de objetos

El reconocimiento de objetos, o clasificación de imágenes, es una técnica de visión computarizada que identifica si un objeto específico está presente o no en una imagen o video (MathWorks, 2018). Esto es posible gracias al entrenamiento de algoritmos de aprendizaje automático, a partir de grandes bases de datos de imágenes y sus etiquetas que indican la presencia del objeto de interés. Su objetivo es obtener información que permita a las máquinas entender el contenido de una imagen. La utilidad de esta información ha logrado que el reconocimiento de objetos se convierta en un recurso clave en vehículos autónomos, la inspección industrial, visión robótica, entre otras aplicaciones variadas.

2.2.1. Técnicas clásicas de clasificación de imágenes

Entre los algoritmos de reconocimiento de objetos que utilizan técnicas de aprendizaje automático clásico, (Ramisa, Aldavert, Vasudevan, Toledo, & Lopez de Mantaras, 2015) mencionan en su libro tres métodos destacados: el algoritmo *SIFT*, la bolsa de características, y clasificadores simples en cascada. Éstos han sido seleccionados en base a su aplicabilidad en la robótica móvil e implementaciones de detección en tiempo real.

El **algoritmo *SIFT*** (del inglés *Scale-Invariant Feature Transform*), o transformación de características de escala invariante, detecta y describe características locales en las imágenes. Las características son detectadas usando cuatro etapas de filtrado (Jafri, Ali, Arabnia, & Fatima, 2014):

1. Se buscan los puntos de interés a partir de una función de diferencia de gaussianos (*DoG*), aplicada en múltiples escalas.
2. Los puntos más resistentes a la distorsión son seleccionados como puntos clave.
3. A cada punto clave se le asigna o más orientaciones, en base a las direcciones de gradientes locales en la imagen.
4. Los gradientes locales de la región cercana a cada punto clave son transformados en representaciones resistentes a distorsión local y cambios de iluminación.

Las características obtenidas son comparadas con las de un conjunto de imágenes de entrenamiento, agrupándolas de acuerdo a su respectiva transformación en traslación, rotación y escala. Una de las principales ventajas que presenta este algoritmo es que no necesita una cantidad excesiva de imágenes para su entrenamiento, debido a su capacidad de relacionar objetos capturados desde distintas perspectivas, como se aprecia en la *Figura 7*.

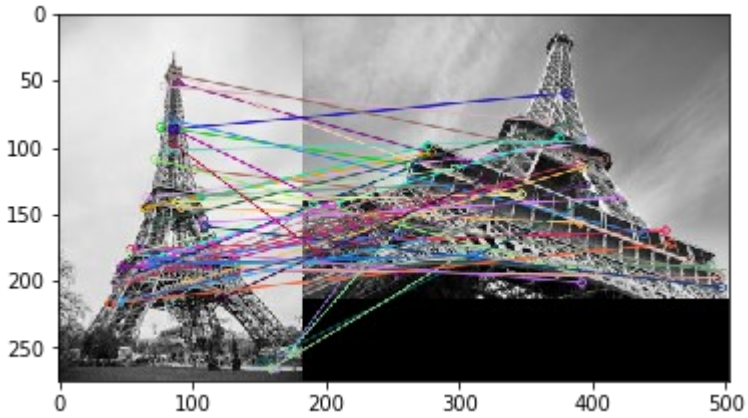


Figura 7. *Coincidencia de características SIFT* (Singh, 2019).

La **bolsa de características** parte del concepto de bolsa de palabras (*bag-of-words*) utilizado en el procesamiento de lenguaje natural, pero adaptado a la clasificación de imágenes. Una bolsa de palabras es una forma de representar un documento o un texto por la frecuencia con la aparecen algunas palabras determinadas (Nistér & Stewénius, 2006). De manera similar, los métodos basados en una bolsa de características, obtienen la representación de una imagen en base a la frecuencia de aparición de ciertos descriptores locales como, por ejemplo, SIFT u otras formas de caracterización de imágenes.

El algoritmo de **clasificadores en cascada** de (Viola & Jones, 2001) es un método diseñado para potenciar la velocidad de ejecución. Varios clasificadores débiles son ensamblados en cascada, usando un algoritmo de aprendizaje basado en Adaboost. Se utilizan representaciones simples como características Haar rectangulares. Cada nodo de la cascada es un filtro que determina la presencia de una única característica Haar en la imagen dada. La construcción de la cascada es realizada mediante aprendizaje automático, de modo que los filtros más discriminativos son asignados a las primeras etapas. Esta configuración permite que el algoritmo pueda identificar rápidamente si el objeto no está presente en la ventana analizada para no emplear más tiempo en ella, y así enfocar los recursos a las secciones más importantes de la imagen.

Otro método muy utilizado, pero que requiere un tiempo mayor de procesamiento, es el **histograma de gradiente orientado *HOG*** (del inglés *Histogram of Oriented Gradient*). Éste representa cada imagen mediante una matriz de histogramas, que acumula las direcciones de gradientes u orientaciones de bordes en secciones pequeñas de la misma. Estos descriptores HOG son usados en un clasificador de máquina de vectores de soporte *SVM* para determinar la presencia de objetos, como humanos (Dalal & Triggs, 2005).

2.2.2. Reconocimiento basado en aprendizaje profundo

Las redes neuronales profundas han revolucionado la manera de dar solución a muchos de los problemas de visión artificial, siendo el reconocimiento y la detección de objetos de los más destacados (Goyal & Benjamin, 2014). Desde el éxito obtenido por *AlexNet* (Krizhevsky, Sutskever, & Hinton, 2012) al utilizar una red neuronal convolucional (CNN) para clasificación de imágenes en el reto *ImageNet* (Russakovsky et al., 2015), las redes convolucionales se convirtieron en el método más utilizado y explorado para el procesamiento de imágenes y videos en el campo de la inteligencia artificial. Una de sus principales ventajas es que la extracción de características es realizada automáticamente dentro de la red neuronal, facilitando su implementación y generalización (Alom et al., 2017).

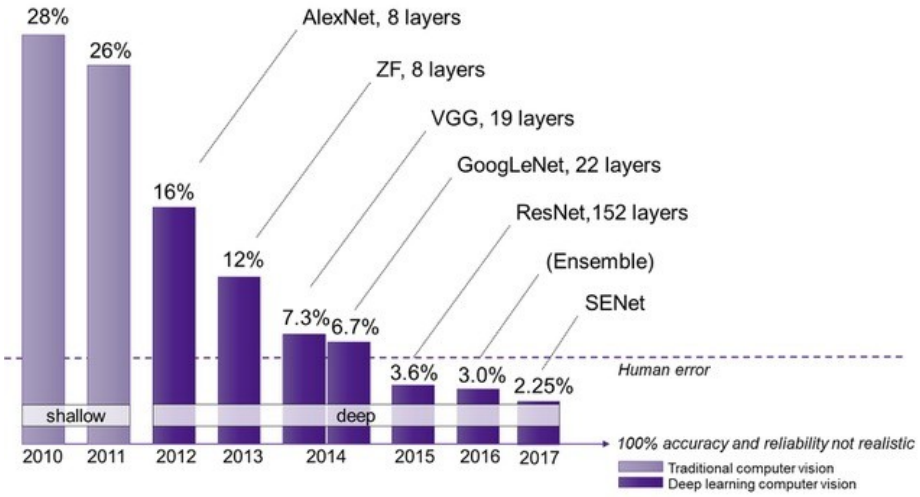


Figura 8. *Resultados del reto ImageNet desde 2010 hasta 2017* (Cooper, 2019).

Entre las arquitecturas de CNN más conocidas, además de *AlexNet*, se encuentran *VGG* (Simonyan & Zisserman, 2015), *GoogLeNet* (Szegedy et al., 2015), *ResNet* (He, Zhang, Ren, & Sun, 2016), y *SENet* (J. Hu, Shen, & Sun, 2018), cuyos resultados en el reto ImageNet se muestran en la *Figura 8*.

2.2.3. Detección de objetos

La detección de objetos combina el reconocimiento de objetos con la identificación de su localización en la imagen. La clasificación de imágenes no proporciona la información suficiente cuando la ubicación de los objetos es un dato de interés, o cuando se quiere identificar a distintos objetos en una misma imagen. Ante la necesidad de resultados más específicos, los algoritmos de reconocimiento de objetos han sido adaptados para convertirse en detectores capaces de diferenciar varios objetos de diferentes clases y la localización de cada uno de ellos, como se puede observar en la *Figura 9*.

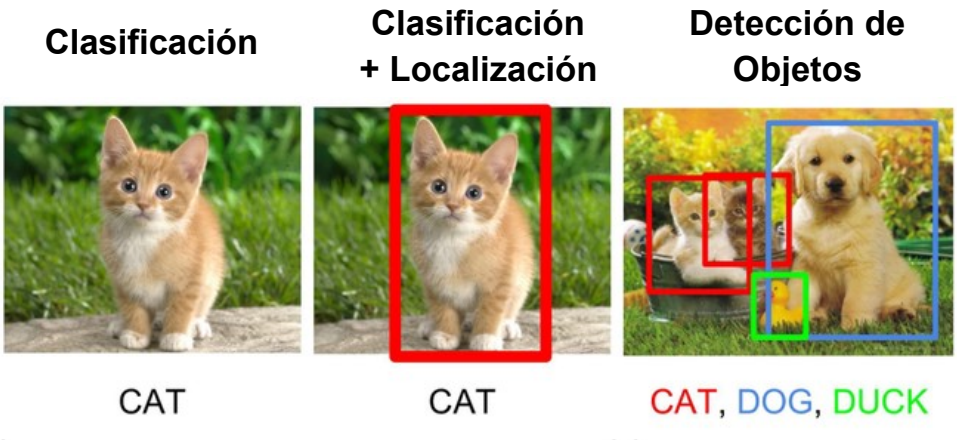


Figura 9. *Clasificación y detección de objetos* (GeeksforGeeks, 2018).

Así como el detector de (Viola & Jones, 2001) mencionado anteriormente, generalmente estos algoritmos realizan una búsqueda por ventanas en la imagen. Al identificar las partes de la imagen que contienen características relacionadas con el objeto buscado, y descartar las partes no relacionadas, la sección de interés es señalada mediante un recuadro. En contraste con los métodos que realizan una búsqueda exhaustiva para la detección, han surgido propuestas distintas, como el trabajo de (Uijlings, Van De Sande, Gevers, & Smeulders, 2013), que utiliza una búsqueda selectiva, guiada por aprendizaje. Su objetivo es reducir el número de localizaciones por analizar, generando un conjunto pequeño de posibles localizaciones de alta calidad.

Por su parte, el sistema de detección *YOLO* (Redmon, Divvala, Girshick, & Farhadi, 2016) trata la detección de objetos como un problema independiente en lugar de manejarlo como una adaptación del reconocimiento. Esta solución utiliza una única red neuronal para obtener la localización de los objetos y la clase a la que pertenecen. La velocidad de ejecución de este algoritmo se destaca, siendo notablemente superior a los enfoques previos, y haciendo de éste una opción relevante para aplicaciones en tiempo real.

Entre otras propuestas recientes que hacen uso de los beneficios del aprendizaje profundo tenemos las redes piramidales de características y las redes de relación para detección de objetos. Las primeras aprovechan jerarquía piramidal y multiescalar inherente de las redes neuronales convolucionales profundas, para estructurar pirámides de características que ayudan en la tarea de detección de objetos a diferentes escalas (Lin et al., 2017). Las redes de relación (H. Hu, Gu, Zhang, Dai, & Wei, 2018) procesan conjuntos de objetos simultáneamente, siendo una solución más completa que evita la necesidad de pasos adicionales como la supresión *non-maxima*.

2.2.4. Detección 3D

Así como la detección de objetos a partir de imágenes 2D ha sido de gran interés y ha tenido mucho éxito en el campo de la visión artificial, la detección tridimensional también ha captado la atención de los investigadores. Se ha incluido información 3D de los objetos en estos algoritmos desde distintas perspectivas.

Antes de la popularización de las redes neuronales para procesamiento de imágenes, se usaban descriptores 3D "manuales", divididos en dos categorías: características globales y características locales. Los métodos globales procesan el objeto como un ente total para el reconocimiento; pero ignoran detalles de forma y requieren una segmentación previa del objeto en la escena. Las características locales extraen solamente superficies locales alrededor de puntos específicos, y permiten una mejor detección de objetos parcialmente ocultos, o en escenas desordenadas (Guo, Bennamoun, Sohel, Lu, & Wan, 2014).

Las redes neuronales convolucionales 2.5D parten de imágenes RGB-D, es decir, agregan la información de profundidad de una imagen 2D como un canal adicional. A pesar de que éstas no son capaces de aprovechar completamente la información geométrica tridimensional (Zhi, Liu, Li, & Guo, 2018), son de gran utilidad para el procesamiento casi directo de imágenes obtenidas a partir de dispositivos como cámaras de profundidad RGB-D o cámaras estéreo.

Finalmente, las redes convoluciones 3D están especialmente diseñadas para tratar con información volumétrica tridimensional, y pueden extraer descriptores de forma 3D altamente discriminativos a partir de ella. Se pueden dividir en dos categorías principales: CNNs de múltiple-vista, y CNNs volumétricas. Las primeras son relativamente bajas en dimensionalidad y en requerimientos computacionales, pero necesita que los objetos se encuentren orientados verticalmente, o conocer la pose de la cámara. Las redes convolucionales volumétricas tienen, en su mayoría, arquitecturas muy grandes y complejas. Esto hace que presenten un buen desempeño en el reconocimiento de objetos, pero a costa de la necesidad de alta capacidad de procesamiento y memoria. Ante esta limitación, *LightNet* (Zhi et al., 2018) surge como propuesta, y se destaca por utilizar aprendizaje multitarea.

Una perspectiva innovadora para mejorar los resultados de detección a partir de imágenes obtenidas por cámaras estéreo es presentada por (Wang et al., 2019), al cambiar su forma de representación para simular los datos obtenidos por sensores LiDAR.

2.2.5. *OpenCV*

OpenCV (del inglés *Open Source Computer Vision Library*) es una librería de código abierto para visión computarizada y aprendizaje automático (OpenCV team, 2020). Está disponible para múltiples plataformas y lenguajes, incluido Python, y es la librería de visión artificial más grande debido a que contiene más de 2500 algoritmos optimizados. Su licencia permite utilizar y modificar el código de forma gratuita para fines comerciales y académicos (Marín, 2020).

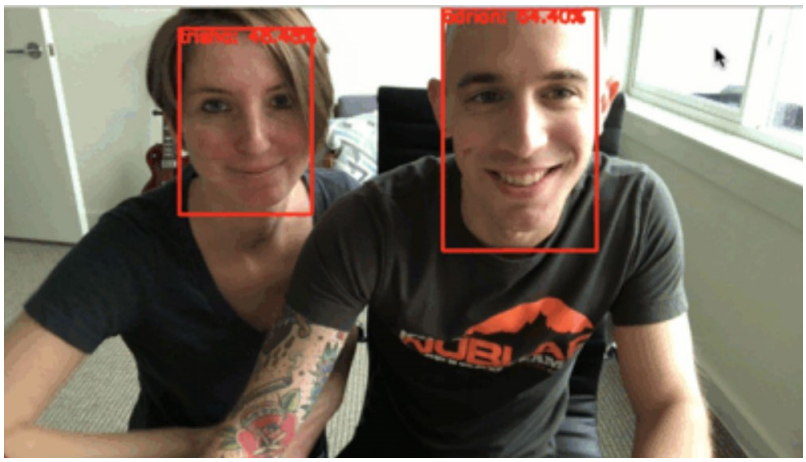


Figura 10. *Reconocimiento de rostros con OpenCV* (Rosebrock, 2018).

Las herramientas que proporciona son de gran utilidad para investigadores y desarrolladores, ya que una gran cantidad de los algoritmos que incluye son muy fáciles de usar. Entre los principales usos que se puede dar a sus funciones están: la detección y reconocimiento de rostros (véase la *Figura 10*), identificación de objetos, seguimiento de objetos móviles, producción de nubes de puntos tridimensionales, etc.

2.3. Navegación autónoma

La navegación autónoma permite que un vehículo o robot móvil sea capaz de planificar su ruta y ejecutarla sin la intervención de un operador humano (Michelson, 2000). Para lograrlo se hace uso de la información de su entorno capturada a través de una gran variedad de sensores, y de los datos almacenados, ya sea localmente o en servidores relacionados. En conjunto con campos relacionados como la inteligencia artificial y la visión computarizada, los algoritmos de navegación autónoma han ganado mucho reconocimiento al lograr poner en movimiento desde pequeños robots y drones, hasta automóviles comerciales (Tesla, 2020) y robots de uso industrial (Mobile Industrial Robots A/S, 2020) como el de la *Figura 11*.

Figura 11. *Robot industrial autónomo para transporte de pallets* (Mobile Industrial Robots A/S, 2020).

2.3.1. *SLAM*

El problema de localización y mapeo simultáneo, más conocido en la literatura como *SLAM* (por sus siglas en inglés *Simultaneous Localization And Mapping*), ha sido abordado desde distintos métodos desde su origen en 1986 (Durrant-Whyte & Bailey, 2006), especialmente por su utilidad en aplicaciones de navegación (Lemaire, Berger, Jung, & Lacroix, 2007) en la robótica móvil y vehículos autónomos. La mayor clasificación de los sistemas SLAM genera dos grupos: métodos fuera de línea (*offline*), y métodos en línea (*online*).

Los algoritmos de *offline SLAM* utilizan la información de los sensores previamente almacenada para realizar un paso de procesado posterior (Frese, Wagner, & Röfer, 2010) cuyo resultado suele ser la construcción de un mapa virtual bidimensional o tridimensional. Estos métodos se caracterizan por obtener buenos valores de exactitud a costa de un alto tiempo de procesamiento necesario, lo que impide que puedan ser utilizados para aplicaciones de tiempo real.

Entre los trabajos más destacados de SLAM fuera de línea encontramos el algoritmo *GraphSLAM* desarrollado en el laboratorio de inteligencia artificial de la Universidad de Stanford (Thrun & Montemerlo, 2006), que sigue la perspectiva tradicional, con un paso reducción innovador que permite la escalabilidad para el mapeo a gran escala de estructuras urbanas, permitiendo que sean manejables ambientes con más de 10^8 características.

En cuanto a los algoritmos de *online SLAM*, su principal característica es que presentan soluciones con un bajo tiempo de procesamiento que les permite ser ejecutadas en tiempo real y en campo. Sin embargo, esta restricción de tiempo es lo que hace que la mayoría de

estos algoritmos no consigan una exactitud muy buena, al presentar un error acumulativo debido a la creciente incertidumbre.

Las implementaciones de SLAM en línea abarcan una gran variedad de sensores para la adquisición de información, y en muchos casos se usan combinaciones de distintos tipos de sensores. Entre los sensores más utilizados se encuentran, además de cámaras RGB, cámaras RGB-D (Mur-Artal & Tardós, 2017), sensores láser (Newman, Cole, & Ho, 2006) y lidars (Kim et al., 2009). Así mismo, entre las combinaciones más populares se destacan las que utilizan cámaras junto con sensores inerciales (Kasyanov & Engelmann, 2017) o cámaras con sensores GPS.

Haciendo énfasis en los sistemas que utilizan cámaras para obtener la información de su entorno, conocidos como *visual-SLAM*, se presentan dos grupos principales: los que utilizan una única cámara para realizar el mapeo del entorno, llamados *SLAM monocular*; y los que utilizan más de una cámara (generalmente dos), conocidos como *SLAM estéreo*.

2.3.1.1. Sistemas monoculares

Los sistemas que utilizan una única entrada de video para realizar la reconstrucción tridimensional del entorno se basan en el procesamiento matemático de los movimientos realizados por la cámara, para calcular la profundidad de los puntos en función al desplazamiento observado en ellos entre capturas tomadas en distintos instantes de tiempo.

Lograr la ejecución de este procesamiento en tiempo real ha sido el principal reto para los investigadores del SLAM monocular. Abordando este reto, Andrew Davison presenta un Sistema Bayesiano *top-down* para la localización de una única cámara (Davison, 2003) a través del mapeo de un conjunto disperso de características, usando el modelamiento del movimiento y una estrategia de medición activa guiada, remarcando la solución del problema de la inicialización de características en tiempo real a través de un muestreo factorizado.

Por otro lado, el algoritmo denominado por sus autores como “odometría visual” (Nistér, Naroditsky, & Bergen, 2004) demuestra buenos resultados en velocidad, latencia, exactitud y robustez al utilizar el esquema de muestreo aleatorio RANSAC predictivo (Nistér, 2005) y su propio desarrollo en el seguimiento de características en tiempo real.

La propuesta de (Mouragnon, Lhuillier, Dhome, Dekeyser, & Sayd, 2006) utiliza los beneficios de los métodos incrementales y con ajuste de paquete fuera de línea, en una aplicación de SLAM en línea para mejorar la exactitud en la reconstrucción tridimensional generada a partir de las imágenes obtenidas por una cámara ubicada en la plataforma móvil.

A pesar de los grandes avances que se realizaron en SLAM monocular, éste ha perdido fuerza durante la última década debido a que los avances de la tecnología han facilitado la utilización de sensores adicionales o sistemas estéreo que mejoran significativamente los resultados.

2.3.1.2. Sistemas estéreo

Los sistemas de SLAM estéreo se caracterizan porque utilizan cámaras estéreo o a su vez dos cámaras individuales sincronizadas para la adquisición de información. Tradicionalmente, la separación entre puntos de interés correspondientes entre ambas imágenes obtenidas es el recurso fundamental para el cómputo de profundidades. Los métodos que abordan el problema de SLAM de esta manera son llamados métodos basados en puntos de interés.

Un ejemplo de éstos es el presentado por (Lategahn, Geiger, & Kitt, 2011), con la particularidad de computar mapas de densidad locales para la navegación de vehículos terrestres autónomos. El sistema desarrollado por (G. Zhang, Lee, Lim, & Suh, 2015) se puede interpretar como una variación innovadora de los métodos basados en puntos de interés, ya que utiliza líneas rectas en lugar de los clásicos puntos de interés como características, demostrando un mejor desempeño de reconstrucción.

A diferencia de los métodos tradicionales, los métodos directos de SLAM no utilizan puntos de interés, sino que se basan en restricciones de foto-consistencia para poder relacionar directamente un gran conjunto de pixeles y así estimar las profundidades para la reconstrucción tridimensional. (Engel, Stückler, & Cremers, 2015) presentan un sistema de SLAM directo a gran escala en tiempo real, utilizando una combinación de estéreo estático (usando imágenes de diferentes cámaras, tomadas en el mismo instante de tiempo) con estéreo temporal (usando imágenes de una misma cámara, tomadas en distintos puntos en el tiempo).

2.3.1.3. *SLAM* + Inteligencia Artificial

La problemática del SLAM no ha quedado fuera de la gran lista de campos explorados por la inteligencia artificial, en especial mediante aprendizaje profundo. La propuesta llamada Neural SLAM (J. Zhang, Tai, Boedecker, Burgard, & Liu, 2017) usa una arquitectura de memoria externa (a largo plazo) junto con una red neuronal profunda completamente diferenciable para para que sea capaz de aprender representaciones del mapa global.

El aprendizaje profundo ha sido también utilizado para entrenar un auto-codificador de mapas de profundidad mediante SLAM monocular (Bloesch, Czarnowski, Clark, Leutenegger, & Davison, 2018), logrando una nueva forma de representación de la geometría del entorno

llamada *codeSLAM*. Así también, Gao y Zhang utilizan las redes neuronales profundas para la detección de contornos cerrados usados como una alternativa para la resolución de problemas de SLAM visual (Gao & Zhang, 2017).

En el artículo de (Milz, Arbeiter, Witt, Abdallah, & Yogamani, 2018) se realiza un análisis de las oportunidades que tiene el aprendizaje profundo para ser usado como reemplazo de las partes que conforman la estructura del SLAM Visual clásico con el fin de mejorar los resultados de los métodos clásicos del estado del arte. Este análisis se realiza en el contexto del uso del SLAM visual para su aplicación en la conducción automatizada.

2.3.2. Detección y evasión de obstáculos

Una de las funciones que debe incluir una máquina para desplazarse de manera autónoma es la capacidad de detectar los obstáculos que se presenten, y evadirlos para no colisionar contra ellos. “La habilidad de los robots móviles para navegar y evadir obstáculos es un indicador importante de la inteligencia del robot” (Liu et al., 2017). Los sistemas de detección de obstáculos han evolucionado notablemente a lo largo de la historia, partiendo desde métodos basados únicamente en sensores simples, hasta los métodos modernos que hacen uso de algoritmos de visión artificial y aprendizaje profundo.

Las primeras propuestas de utilizar redes neuronales profundas para la detección de obstáculos surgen poco después de la popularización de las redes convolucionales para clasificación de imágenes, como el trabajo de (Yu, Hong, Huang, & Wang, 2013). Algunos años más tarde aparecen propuestas más completas que incluyen instrucciones para evadir los obstáculos detectados (Liu et al., 2017). Otros trabajos similares, enfocados en la detección de obstáculos para conducción autónoma son los de (Deepika & Sajith Variyar, 2017) y (Prabhakar, Kailath, Natarajan, & Kumar, 2017).

Los algoritmos para detección y evasión de obstáculos no se han limitado únicamente a vehículos terrestres, sino que también han sido utilizados en drones aéreos o *UAVs*. La propuesta de (Levkovits-Scherer, Cruz-Vega, & Martinez-Carranza, 2019) parte de la red CNN pre-entrenada *MobileNet* para, mediante transferencia de aprendizaje adaptarla para evitar colisiones en tiempo real en un *UAV* con visión monocular.

2.4. Manipulación de objetos y obstáculos

En los tiempos modernos, los robots se desempeñan cada vez mejor en ambientes no estructurados, sin estar limitados a una celda de trabajo estática o entornos diseñados específicamente para agentes robóticos. Al realizar sus tareas asignadas en medios poco controlados, es necesario responder adecuadamente ante las perturbaciones o anormalidades que puedan surgir, como la presencia de obstáculos. Frente a esta necesidad, se han desarrollado propuestas para, por ejemplo, detectar y hacer a un lado los obstáculos que impiden a un brazo robótico alcanzar su objetivo.

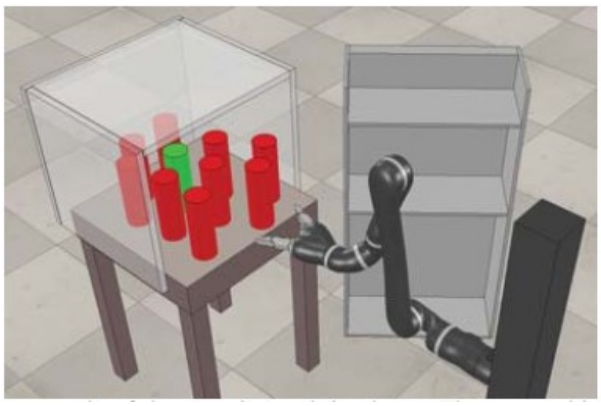


Figura 12. *Reubicación de obstáculos* (Lee, Cho, Nam, Park, & Kim, 2019).

Trabajos como el de (Zhao et al., 2018) detectan los obstáculos presentes en el camino entre el manipulador robótico y su objetivo, y planifican automáticamente las acciones que deben ejecutar. De ser el caso, el robot moverá los obstáculos a una nueva posición que le permita alcanzar el objeto deseado sin colisionar contra ellos. La cantidad de obstáculos puede ser grande, como se observa en la *Figura 12*, por lo que el algoritmo de (Lee et al., 2019) procura minimizar el número de obstáculos que deben ser reubicados, con el fin de optimizar el tiempo de ejecución.

Así como la manipulación de obstáculos puede llegar a ser una característica necesaria para realizar satisfactoriamente las metas de un manipulador robótico industrial, también puede representar una gran ventaja en el campo de la robótica móvil en ambientes con altas limitaciones de movilidad. Sin embargo, hasta la actualidad, la manipulación de obstáculos por robots móviles es un área prácticamente inexplorada. El trabajo que más se acerca a combinar estos campos es el de (Li & Xiong, 2019), ya que incluye la evasión de obstáculos mientras un robot móvil se encuentra realizando una actividad de manipulación; mas no trata con la problemática de la manipulación de obstáculos.

2.5. Resumen

Mientras que la evasión de obstáculos ha sido ampliamente explorada en el campo de la navegación autónoma, se encuentran pocos aportes respecto a la posibilidad de ejecutar acciones alternativas como manipularlos.

A pesar de que muchos de los robots móviles autónomos pueden detectar y reconocer objetos de distintas categorías en su entorno, son incapaces de modificar su comportamiento ante diferentes tipos de obstáculos. La ausencia de esta característica no resulta significativa durante la movilización en espacios abiertos, pero puede llegar a ser muy notoria en entornos reducidos, limitados por puertas y pasillos estrechos, y con alta probabilidad de interacción con personas o animales, así como con objetos varios que podrían encontrarse sobre el suelo.

La contribución de este trabajo radica en proveer al robot la capacidad de identificar si un obstáculo es manipulable o no. La posibilidad de manipular objetos que no representen riesgo, cuando estén obstaculizando la ruta de navegación, dota al agente una mayor flexibilidad para continuar con sus tareas. Esta propuesta consiste de tres fases principales: detección, clasificación, y toma de decisiones. Mediante mapas de profundidad generados a partir de visión estéreo, se detecta la presencia de obstáculos en un rango relevante de profundidades y se obtiene la localización de éstos en la imagen. Se utiliza una red neuronal convolucional profunda para clasificar los obstáculos encontrados en dos categorías: común y crítico. Finalmente, según el espacio disponible estimado y el tipo de obstáculos, se identificará la acción más adecuada, como evadir o manipular el obstáculo de acuerdo a su naturaleza.

3. Objetivos y metodología de trabajo

3.1. Objetivo general

Desarrollar un software de detección y clasificación de obstáculos cercanos mediante algoritmos de visión artificial, orientado a la toma de decisiones que protejan la integridad física de niños y mascotas que se encuentren en el entorno de robots autónomos, utilizando técnicas de aprendizaje profundo y mapas de profundidad generados por una cámara estéreo.

3.2. Objetivos específicos

- Realizar un análisis del estado del arte en sistemas de reconocimiento de obstáculos para robots de navegación autónoma y visión estéreo.
- Localizar los obstáculos que se encuentren más cercanos a las cámaras e identificar si existe el espacio suficiente para evadirlos, a partir de un mapa de profundidades.
- Clasificar los obstáculos detectados en común o crítico, mediante el entrenamiento de una red neuronal convolucional profunda.
- Tomar decisiones inteligentes que determinen las acciones que deben ejecutarse de manera adecuada y segura, en base a toda la información adquirida.
- Implementar el sistema en una tarjeta *StereoPi* y evaluar los resultados obtenidos ubicándola en varios escenarios de prueba.

3.3. Metodología del trabajo

Con el fin de alcanzar los objetivos del presente desarrollo, se utilizará una de las metodologías más utilizadas para proyectos de inteligencia artificial, llamada CRISP-DM por sus siglas en inglés para “CRoss-Industry Standard Process for Data Mining”. El proceso que se indica en la CRISP-DM consta de 6 fases que se muestran en la *Figura 13*. La descripción de dichas fases y cómo han sido adaptadas a este proyecto se detalla en los siguientes apartados.

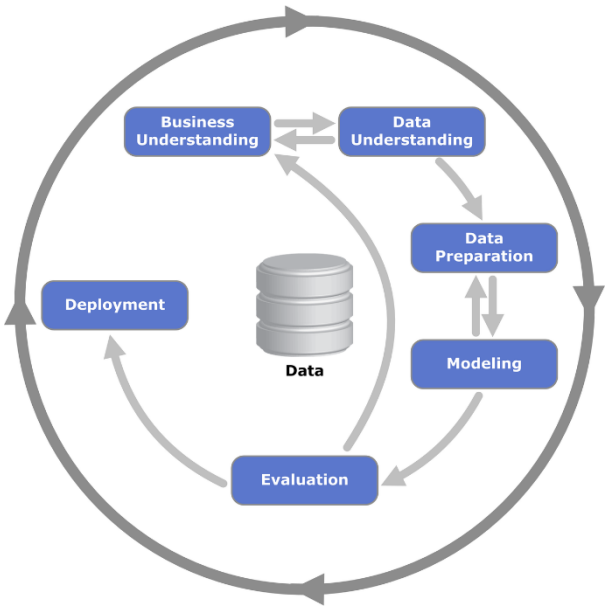


Figura 13. *Fases de la metodología CRISP-DM* (Dadouche, 2018).

3.3.1. Entendimiento del tema

La primera etapa de la metodología CRISP-DM consiste en el entendimiento del tema, estableciendo adecuadamente los objetivos del proyecto en base a lo que se busca alcanzar e identificando los requerimientos del desarrollo. En base al estudio del estado del arte en el tema de interés, se han establecido los objetivos y requisitos para este proyecto, que se encuentran en las secciones *3.1.* Objetivo general, *3.2.* Objetivos específicos y *4.* Identificación de requisitos.

3.3.2. Entendimiento de la información

La etapa de entendimiento de los datos empieza por la recolección de datos que contengan la información necesaria y disponible para la ejecución del proyecto. Estos datos suelen provenir de bases de datos públicas que se pueden encontrar en internet, pero también pueden ser datos propios si se dispone de la suficiente cantidad de información. Es fundamental realizar un análisis correcto de esta información para identificar adecuadamente los pasos siguientes correspondientes al preprocesamiento y para escoger los algoritmos que presenten mejores resultados al tratar el tipo de datos que se van a utilizar.

Este proyecto utiliza un modelo de clasificación entrenado a partir de imágenes de varias bases de datos y fotografías propias. El entendimiento de la información incluye el análisis de estas imágenes de entrenamiento. Adicionalmente, en este caso es de gran importancia

analizar con detalle la información obtenida en la etapa de captura para identificar las siguientes operaciones a realizar. La sección *5.1. Captura y análisis de datos* describe las herramientas de hardware y software utilizadas para percibir el entorno y las características de los datos obtenidos; y la sección *5.4.1. Selección y análisis de bases de datos* detalla el tipo de información contenida en las bases de datos utilizadas para el entrenamiento.

3.3.3. Preparación de datos

La etapa de preparación de los datos consiste en realizar las operaciones necesarias a la información obtenida inicialmente para que sea útil y entendible por los algoritmos utilizados en la etapa de procesamiento. Esta etapa se detalla en tres secciones. Las secciones *5.2. Calibración de la cámara estéreo* y *5.3. Ajustes para el mapa de profundidades* describen las operaciones de configuración de parámetros que deben ejecutarse previamente a la puesta en marcha de este tipo de sistemas. La sección *5.4.2. Preprocesamiento de datos* describe las operaciones de preprocesamiento realizadas en las imágenes previo al entrenamiento del modelo de aprendizaje profundo.

3.3.4. Modelamiento

La etapa de modelamiento describe las operaciones realizadas durante el procesamiento específico de la información ya tratada, y el entrenamiento del modelo de clasificación. Se debe realizar una adecuada selección de las técnicas y algoritmos a utilizar, de acuerdo al tipo de información que se dispone y los resultados que se buscan. La parte principal de esta etapa se detalla en la sección *5.4.3. Entrenamiento del modelo*. Sin embargo, parte fundamental del software desarrollado consiste en el procesamiento descrito en las secciones *5.5. Detección de obstáculos* y *5.7. Módulo de decisiones*. El conjunto de todos estos procedimientos permite obtener los resultados requeridos.

3.3.5. Evaluación

En la etapa de evaluación se llevan a cabo las pruebas necesarias para determinar en qué grado los resultados obtenidos por la etapa de procesamiento son satisfactorios o no. Una evaluación inicial de la red neuronal modelada consta en la sección *5.4.4. Evaluación del modelo entrenado*. Finalmente, se llevaron a cabo pruebas en un entorno real para observar el comportamiento del sistema en implementaciones de campo. El proceso de ejecución de estas pruebas y los resultados obtenidos a partir de ellas se describe en la sección *6. Pruebas y resultados.*

3.3.6. Producción

Como parte de la etapa de producción se encuentra el guardado y exportado de la red neuronal entrenada, descrita en la sección *5.4.5. Serialización del modelo*. Para finalizar, la última etapa de este proyecto consiste en la elaboración de este artículo como memoria del proceso involucrado en el desarrollo de la herramienta de software, sus pruebas y conclusiones.

4. Identificación de requisitos

La primera fase del proyecto consiste en el entendimiento del problema, y con esto es fundamental identificar los requisitos de software para el desarrollo de la herramienta y de hardware para llevar a cabo las pruebas necesarias. El problema que este proyecto trata es la toma de decisiones basada en la detección y reconocimiento de obstáculos, para proteger la integridad física de los niños o mascotas (llamados *objetos críticos*) que se encuentren en el entorno de robots autónomos.

Tabla 1. Tabla de requisitos

Función	**Resultado**	**Ejemplo**
Conocimiento previo	- Dimensiones del robot	Anchura: 50 cm
Estimación del espacio disponible (parcial y total)	Valor numérico continuo, clasificado en: - Suficiente - Insuficiente	- Espacio = 85 cm. Clasificación: Suficiente - Espacio = 40 cm. Clasificación: Insuficiente
Detección de obstáculos	- Obstáculo detectado - Obstáculo no detectado	- Existe obstáculo - No existe obstáculo
Identificación de obstáculos	- Común - Crítico	- Juguete / objeto pequeño - Mascota / niño
Toma de decisiones	- Continuar trayectoria - Evadir obstáculo - Acción crítica: manipular obstáculo - Cambio de ruta	De acuerdo a la *Tabla 3*

Las funciones y resultados requeridos se muestran en la *Tabla 1* y son los siguientes. Un algoritmo de detección de obstáculos cercanos, en conjunto con un modelo de clasificación entrenado con algoritmos de aprendizaje profundo, deberá ser capaz de identificar la presencia de objetos críticos en la ruta del robot. Esto puede ser logrado mediante un clasificador binario que etiquete al obstáculo en una de dos categorías posibles: obstáculo común, u obstáculo crítico. Al mismo tiempo, se debe identificar la distancia existente entre el obstáculo y la pared más alejada, utilizando el mapa de disparidades obtenido mediante la visión estéreo, para conocer si el espacio disponible es suficiente para que el robot pueda evadir el obstáculo. Los algoritmos y técnicas necesarios se muestran en la *Tabla 2*.

Tabla 2. Tabla de técnicas a utilizar

Etapa	**Algoritmo / técnica**	**Fuente**
Generación del mapa de profundidades	*Stereo block matching SBM*	OpenCV
Localización de obstáculos	Búsqueda exhaustiva en una zona determinada del mapa de profundidades	Autoría propia
Reconocimiento de objetos	Entrenamiento de una red neuronal convolucional (aprendizaje profundo)	Autoría propia
Toma de decisiones	Secuencia de condiciones lógicas	Autoría propia

Finalmente, una etapa de toma de decisiones deberá utilizar toda la información obtenida para elegir la mejor acción a ejecutar. Por ejemplo: si se detecta un obstáculo crítico, pero existe suficiente espacio en la ruta para su evasión por un costado, se procederá a realizar ligeros cambios en la ruta trazada para pasar por el espacio disponible. Si existe un obstáculo común, y no se dispone del espacio suficiente para evadirlo, se podrá considerar la posibilidad de manipularlo para poder continuar en la trayectoria actual. La decisión específica para cada combinación posible de condiciones se detalla en la *Tabla 3*.

Para la captura de las imágenes estéreo se utilizará la tarjeta *StereoPi*, que es una cámara estereoscópica de código abierto basada en *RaspberryPi* conformada por dos sensores de 5 MP (eLinux, 2018). Se utilizará el sistema operativo *Raspbian Stretch* junto con herramientas que facilitan operaciones de visión computarizada como *OpenCV* (OpenCV team, 2020). Si se necesitan velocidades altas de procesamiento, éste puede ser realizado por un ordenador externo con mayor capacidad de cómputo conectado a la tarjeta; pero por facilidad, en este trabajo todo el procesamiento será realizado por la *StereoPi.*

Tabla 3. Tabla de decisiones

Estado	**Decisión resultante**
- Obstáculo no detectado. - Espacio parcial = espacio total: Suficiente.	Continuar trayectoria
- Obstáculo no detectado - Espacio parcial = espacio total: Insuficiente	Cambio de ruta
- Obstáculo detectado. - Espacio parcial: Suficiente. Espacio total: Suficiente.	Evadir obstáculo y continuar
- Obstáculo detectado. - Espacio parcial: Insuficiente. Espacio total: Suficiente. - Tipo de obstáculo: Crítico	Emitir alerta sonora y esperar a que el obstáculo se mueva; caso contrario, cambiar de ruta.
- Obstáculo detectado. - Espacio parcial: Insuficiente. Espacio total: Suficiente. - Tipo de obstáculo: Común	Acción alternativa: Manipular obstáculo para poder continuar
- Obstáculo detectado. - Espacio parcial: Insuficiente. Espacio total: Insuficiente.	Cambio de ruta

5. Descripción de la herramienta software desarrollada

La contribución presentada consiste en un software de identificación y clasificación de obstáculos, orientado a la toma de decisiones para robots de navegación autónoma en entornos de espacio reducido. El desarrollo ha sido pensado para su implementación en tarjetas basadas en *Raspberry Pi*, debido a que son altamente usadas en aplicaciones de investigación en robótica. Se utilizan dos cámaras a modo de cámara estereoscópica para el reconocimiento del tipo de obstáculo y la estimación de profundidades. Los datos de profundidad permiten al robot identificar los obstáculos cercanos, y saber si existe el espacio necesario para evadir el obstáculo; caso contrario, clasificarlo para conocer si es posible manipularlo para continuar su navegación. Esta herramienta desarrollada en Python hace uso de algunas funciones proporcionadas por la librería *OpenCV* y una red neuronal convolucional profunda para clasificar los obstáculos en dos tipos: comunes y críticos, siendo los críticos personas, principalmente niños pequeños debido al rango de visibilidad del hardware, o mascotas. De esta forma, el robot podrá reconocer si el obstáculo es manipulable (común) o no manipulable (crítico), para “decidir” su siguiente paso.

5.1. Captura y análisis de datos

La información de entrada consiste en imágenes RGB estéreo con resolución de 1280x480 píxeles, obtenidas por una cámara estereoscópica casera, construida con los elementos incluidos en el paquete inicial (*Starter Kit*) de *StereoPi*, mostrada en la *Figura 14*. Ésta se compone de dos cámaras para *Raspberry Pi* de 5 megapíxeles, ubicadas con una separación horizontal de 6.5 cm y orientadas en la misma dirección.

Figura 14. “*StereoPi Starter Kit” montado para este trabajo*.

Las imágenes estéreo capturadas por la *StereoPi* corresponden a fotografías del entorno, donde se observan principalmente paredes, obstáculos y objetos varios alejados del robot a distancias entre 0.5 y 4 metros. La *Tabla 4* detalla las características de la tarjeta *StereoPi* utilizada para la captura de imágenes estéreo.

Tabla 4. Características técnicas de la tarjeta *StereoPi* utilizada

RaspberryPi Compute Module 3+ Lite	• Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.2GHz • SDRAM LPDDR2 1GB
2 Cámaras para *RaspberryPi* V1 (OV5647 sensor)	• OmniVision OV5647 Color CMOS QSXGA (5-megapixel) • Tamaño del sensor: 3.67 x 2.74 mm • Número de pixeles: 2592 x 1944 • Tamaño de pixel: 1.4 x 1.4 um • Lente: f=3.6 mm, f/2.9 • Ángulo de visión: 54 x 41 grados • Línea base entre cámaras: 6.5 cm
Almacenamiento	• Ranura para tarjetas MicroSD • Tarjeta MicroSD usada: 16GB
Alimentación	• 5 VDC mediante conector de 2 pines, con cable USB adaptado
Salida de video	• HDMI
Conexión de red	• USB: 2 x USB Tipo-A • Ethernet: conector RJ45

Información obtenida de *StereoPi Wiki* (virt2real, 2019b)

La lectura de las imágenes se realiza mediante el uso de la librería *"picamera"* (Jones, 2013), especializada para el uso de módulos de cámara junto con tarjetas *Raspberry Pi*. Se obtiene una captura continua de imágenes estéreo a través del puerto de video. En la configuración lado-a-lado del modo estéreo, cada cuadro obtenido es una imagen a color que contiene la vista de la cámara izquierda y la vista de la cámara derecha ubicadas una al lado de la otra. Por esta razón, una resolución de 1280x480 significa que la imagen estéreo está compuesta por dos imágenes de 640x480 píxeles, siendo esta última la resolución individual de las cámaras.

La *StereoPi* utiliza la instalación de la imagen de disco "*Raspbian Stretch OpenCV image*" proporcionada en la página web *StereoPi Wiki* (virt2real, 2020), la cual está basada en el sistema operativo Raspbian Stretch (9) y contiene la librería *OpenCV* pre-instalada.

5.2. Calibración de la cámara estéreo

Cada cámara cuenta con distintos parámetros internos y externos. Además, al utilizar una cámara estéreo es imposible alinear las cámaras perfectamente. Por estas razones, es necesario utilizar un método de calibración por software que utiliza fotografías de un patrón de ajedrez para encontrar los parámetros adecuados que permitan rectificar las imágenes estéreo. Se realizó el proceso de calibración descrito en el blog de *StereoPi* (virt2real, 2019a), utilizando un patrón de ajedrez 7x9 impreso en una hoja de tamaño A4, donde la longitud de cada lado de los cuadros es de 2 cm. La *Figura 15* muestra una de las capturas utilizadas para la calibración de las cámaras.

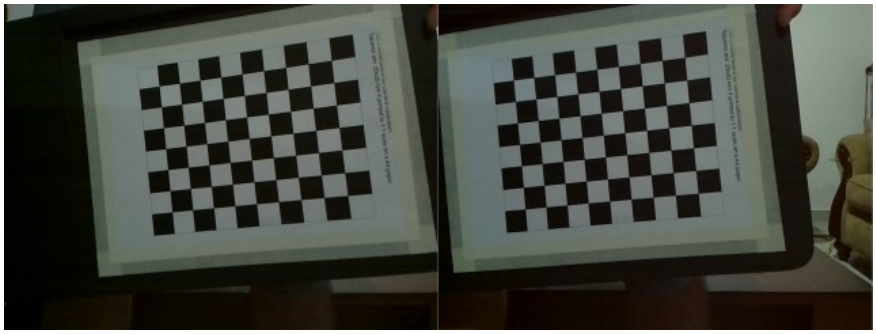


Figura 15. Captura estéreo del patrón de ajedrez para calibración.

A partir de los parámetros obtenidos durante el proceso de calibración estéreo, se ejecuta la rectificación de las imágenes, cuyo resultado se muestra en la *Figura 16*. La rectificación alinea las dos imágenes que conforman una captura estéreo, con el fin de facilitar la búsqueda de correspondencia entre píxeles o bloques. Encontrar los correctos pares de píxeles correspondientes permitirá calcular de forma adecuada las disparidades y, por lo tanto, generar un mapa de disparidad o de profundidad óptimo.

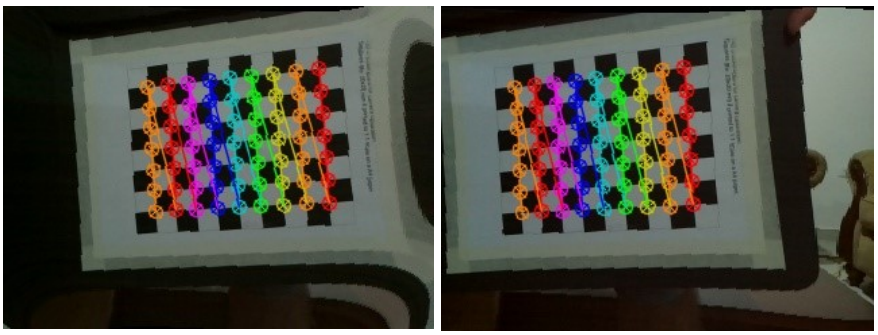


Figura 16. Imágenes rectificadas después de la calibración.

5.3. Ajustes para el mapa de profundidades

Los parámetros utilizados para la generación del mapa de profundidades también deben ser configurados correctamente para obtener los resultados deseados. El software proporcionado en el *tutorial de OpenCV y mapa de profundidades en StereoPi* (virt2real, 2019a) permite ajustar estos parámetros mientras se observa gráficamente el mapa generado con cada cambio de ajustes.

Estos ajustes son guardados en un archivo de texto llamado *"3dmap_set.txt"* al presionar el botón *Save to file*. Los parámetros disponibles y sus valores utilizados son:

- *SADWindowSize*: 5
- *minDisparity*: -20
- *numberOfDisparities*: 80
- *preFilterCap*: 29
- *preFilterSize*: 5
- *speckleRange*: 4
- *speckleWindowSize*: 100
- *textureThreshold*: 200
- *uniquenessRatio*: 3

La ventana de configuración que muestra los resultados obtenidos con los parámetros antes detallados se muestra en la *Figura 17*.

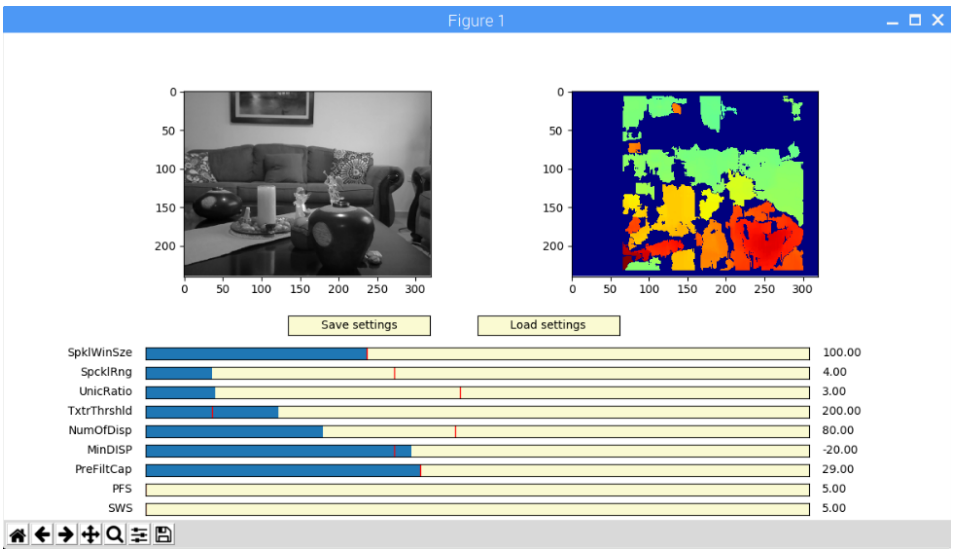


Figura 17. Ventana de ajuste de parámetros para el mapa de profundidades.

La imagen estéreo utilizada como referencia para ajustar los parámetros en este trabajo es la que se muestra en la *Figura 18*.

Figura 18. Imagen de referencia para el ajuste de parámetros.

Seleccionar los valores correctos de estos parámetros es fundamental para detectar el rango deseado de profundidades, emparejar la mayor cantidad de puntos posibles y minimizar el número de errores o ruido.

5.4. Entrenamiento de la red neuronal para clasificación

Se realizó el entrenamiento de un modelo de aprendizaje profundo para el reconocimiento de los obstáculos a partir de imágenes. Este modelo consiste de una red neuronal convolucional de clasificación binaria de los obstáculos en dos categorías: comunes, o críticos. Los obstáculos comunes son artículos variados que podrían encontrarse en el camino y podrían ser manipulados sin representar riesgos, como juguetes, calzado, útiles escolares o de oficina, herramientas, entre otros. Los obstáculos críticos son personas como bebés o niños pequeños, y animales domésticos como perros o gatos. Una interacción demasiado cercana entre éstos y el robot podría ser riesgosa, por lo que se tomarán acciones que no representen peligro alguno.

5.4.1. Selección y análisis de bases de datos

Las imágenes utilizadas para el entrenamiento han sido obtenidas de distintas bases de datos junto con imágenes propias y se han seleccionado las más relevantes de acuerdo a los obstáculos que se pueden encontrar en entornos cerrados. Éstas corresponden a fotografías de objetos variados que han sido agrupadas en las dos categorías de interés para este trabajo: comunes o críticos.

Las imágenes de mascotas se obtuvieron de la base de datos *The Oxford-IIIT Pet Dataset* presentada por (Parkhi, Vedaldi, Zisserman, & Jawahar, 2012). La base de datos completa

contiene 4978 imágenes de perros 2371 imágenes de gatos de 37 razas, de las cuales se utilizó una muestra aleatoria de 2000 imágenes unificadas en una sola categoría. Se utilizaron las miniaturas de las primeras 1000 imágenes de rostros de la base de datos *Flickr-Faces-HQ Dataset* (Karras, Laine, & Aila, 2019), que contienen fotografías de rostros de personas de distintas edades y razas, en resolución 128x128 pixeles. Adicionalmente, se añadieron a los datos de entrenamiento 42 imágenes de bebés obtenidas por medio de la búsqueda en *Google* y 98 fotografías propias de los objetos utilizados como obstáculos críticos en las pruebas, desde distintas perspectivas.

Las imágenes de obstáculos comunes fueron seleccionadas de distintas categorías de la base de datos *Caltech 256* (Griffin, Holub, & Perona, 2006). Se utilizaron 2000 imágenes, entre las cuales se pueden encontrar pelotas de distintos tipos, sombreros, zapatos, botellas, yoyos, relojes, vehículos, entre otros. Se añadieron 55 fotografías propias de algunos objetos utilizados como obstáculos críticos y de distintos ambientes del hogar utilizados como escenarios de prueba.

En total se compilaron 3140 imágenes de la categoría “obstáculos críticos”, y 2055 imágenes de la categoría “obstáculos comunes”, según el detalle en la *Tabla 5*. De éstas, 4000 fueron utilizadas para el entrenamiento del modelo de clasificación y 1195 para su evaluación.

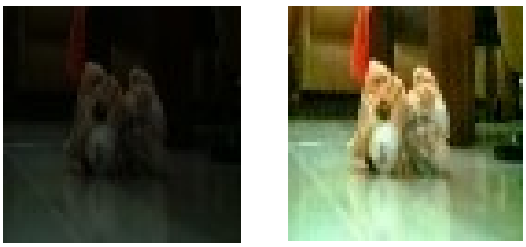
Tabla 5. Imágenes utilizadas para el entrenamiento del modelo de clasificación CNN

Tipo de obstáculo	***Número de imágenes***	***Base de datos de origen***	***Contenido de las imágenes***	***Resolución [píxeles]***
1: Crítico	2000	*The Oxford-IIIT Pet Dataset*	Perros y gatos	Variada
	1000	*Flickr-Faces-HQ Dataset*	Rostros de personas	128x128
	42	*Google*	Bebés	Variada
	98	*Fotografías propias*	Objetos críticos de prueba	4000x3000 3000x3000
0: Común	2000	*Caltech 256*	Objetos variados	Variada
	55	*Fotografías propias*	Objetos comunes de prueba	3000x3000
Total	**5195**			

5.4.2. Preprocesamiento de datos

Las imágenes recopiladas de las distintas bases de datos deben ser preparadas y normalizadas para un entrenamiento adecuado de la red neuronal. La primera operación de preprocesamiento es el ajuste de dimensiones. Las imágenes utilizadas tienen dimensiones distintas, por lo que han sido redimensionadas a un tamaño estándar de 64 x 64 píxeles.

Como siguiente paso, se realiza una ecualización de histograma para realzar el contraste de las imágenes. Debido a que no es posible llevar a cabo esta operación directamente sobre imágenes RGB, éstas son primero transformadas al modelo HSV (Matiz, Saturación, Brillo; del inglés: *Hue, Saturation, Brightness*) para realizar la ecualización de histograma sobre el canal correspondiente al brillo. Posteriormente, la imagen resultante es transformada nuevamente al modelo RGB. Esta operación mejora la visualización y entendimiento de las imágenes y capturas oscuras que inicialmente dificultan la identificación de los objetos que se encuentran en ellas, como se observa en la *Figura 19*.



a) b)

Figura 19. Ecualización de histograma. a) Imagen original.
b) Imagen con realce de contraste.

Finalmente, las imágenes son normalizadas a números decimales en un rango entre 0 y 1 mediante la división de los valores de todos sus canales entre 255.

El tipo de objeto correspondiente a cada imagen fue asignado a una variable categórica de longitud 2, donde el índice 0 corresponde a la clase “obstáculo común”, y el índice 1 corresponde a la clase “obstáculo crítico”.

5.4.3. Entrenamiento del modelo

El entrenamiento del modelo de aprendizaje profundo fue realizado en la interfaz *Keras* para *python* utilizando *TensorFlow* como backend. Se generó una red neuronal convolucional de aproximadamente 10 millones de parámetros con la arquitectura detallada en la *Figura 20*.

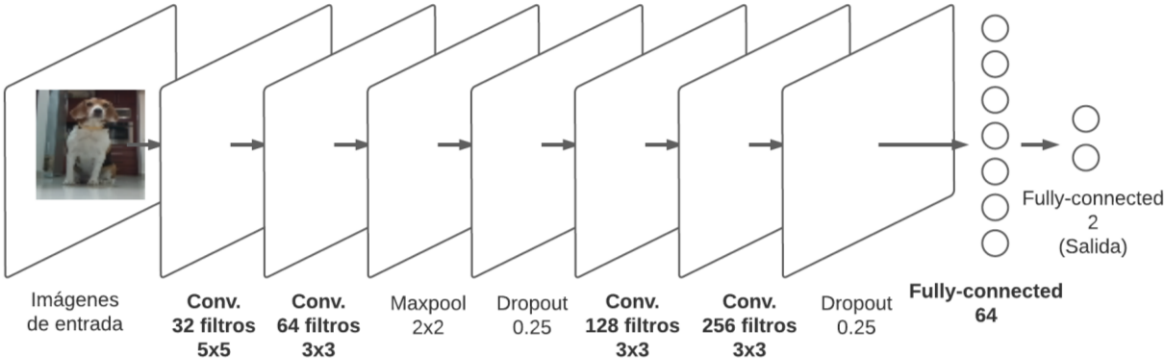
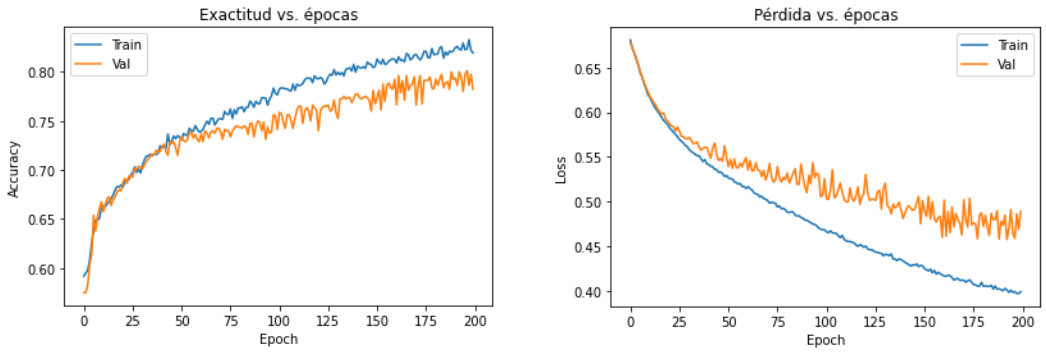


Figura 20. Arquitectura de la red neuronal convolucional entrenada.

El modelo está compuesto por 4 capas convolucionales de 32, 64, 128, y 256 filtros, correspondientemente. El tamaño de filtros de la primera de estas capas es 5x5, mientras que las demás poseen todas filtros de dimensiones 3x3. Las 2 capas finales son capas densas, una de 64 neuronas y la capa de salida con 2 neuronas, de acuerdo al número de clases. La función de activación para todas las capas ocultas es la unidad lineal rectificada *ReLU*. Debido a que la salida corresponde a una variable categórica, la capa final utiliza la función de activación *Softmax*.

Para la regularización de la red se utilizó una capa *MaxPooling* después de la segunda capa convolucional y dos capas *Dropout* ubicadas después de la capa MaxPooling y después de la última capa convolucional, respectivamente.

La función de pérdida más adecuada para problemas de clasificación con salida de tipo categórica es la función de entropía categórica cruzada (*categorical cross-entropy*). Se utilizó esta función de pérdida con el optimizador *AdaDelta* para el entrenamiento del modelo en 200 épocas con 64 imágenes por lote (*batch*). El 20% de los datos de entrenamiento fueron utilizados como conjunto de validación durante el entrenamiento.



a) b)

Figura 21. Evolución de las métricas durante el entrenamiento.
a) Pérdida vs. épocas. b) Exactitud vs. épocas.

La *Figura 21* muestra la evolución de los valores de pérdida y de exactitud en los subconjuntos de entrenamiento y validación, según el número de épocas durante el entrenamiento de la red neuronal.

5.4.4. Evaluación del modelo entrenado

Al aplicar la red neuronal entrenada sobre el conjunto de datos de evaluación, se obtuvo un valor de pérdida de 0.42 y una exactitud de 79.9%. A pesar de no ser los valores óptimos deseados, son resultados buenos debido a que la capacidad de procesamiento la *StereoPi* utilizada para la implementación dificulta el uso de modelos más complejos. En la *Tabla 6* se muestra la matriz de confusión generada a partir de los resultados obtenidos al aplicar el modelo entrenado sobre los datos de evaluación.

Tabla 6. Matriz de confusión

Clase real	***Clase predicha: Objetos comunes***	***Clase predicha: Objetos críticos***	***Total***
Objetos comunes	320	152	472
Objetos críticos	97	623	710
Total	417	775	

5.4.5. Serialización del modelo

Finalmente, el grafo de la red neuronal ya entrenada fue congelado y guardado, para de esta manera poder exportar el modelo completo al formato *protobuf* (.pb), incluyendo su arquitectura y sus pesos. Este archivo permitirá la utilización a futuro del modelo de clasificación desde la *StereoPi* mediante las funciones incluidas en la librería *OpenCV*, sin la necesidad de realizar en ella la instalación de *TensorFlow*.

5.5. Detección de obstáculos

Las imágenes capturadas por la *StereoPi* son preparadas para poder ser procesadas por el algoritmo generador de mapas de profundidad, mediante las siguientes operaciones de preprocesamiento: escalado y separación, conversión a escala de grises, y rectificación. Una vez que las imágenes han sido preparadas adecuadamente, se procede al procesamiento de las mismas para obtener el mapa de profundidades, identificar y estimar el espacio disponible en la ruta crítica, y localizar los obstáculos existentes.

5.5.1. Escalado y separación

El tiempo de procesamiento de cada cuadro capturado está altamente relacionado con las dimensiones de las imágenes utilizadas, especialmente cuando se utilizan dispositivos con una capacidad de cómputo relativamente baja como la *StereoPi*. Al trabajar con aplicaciones en tiempo real se busca obtener la mayor velocidad posible, y para lograrlo se debe usar la menor resolución disponible que permita alcanzar los resultados esperados.

Por dicha razón, cada imagen estéreo obtenida es escalada con un factor de 0.5 antes de continuar con la ejecución de las demás operaciones. La resolución de la imagen resultante es de 640x240 píxeles después del escalado. Como otra alternativa, se pudo haber obtenido la imagen con esta resolución desde el principio; sin embargo, al hacerlo de esa manera la imagen presentaba una distorsión producida por posibles errores en la librería utilizada.

a) b)

Figura 22. Captura estéreo escalada y separada en dos imágenes con resolución 320x240. a) Vista izquierda. b) Vista derecha.

Una vez que ha aplicado el escalado, la captura estéreo debe ser separada en las dos imágenes individuales que la conforman. El resultado de esta separación son dos imágenes de resolución 320x240 píxeles cada una, que llamaremos *"vista izquierda"* y *"vista derecha"* por su naturaleza, como se ilustra en la *Figura 22*.

5.5.2. Conversión a escala de grises

Con el fin de agilizar las siguientes operaciones, las imágenes a color son convertidas a escala de grises. Esta operación es realizada mediante el uso de la función *cvtColor* y el parámetro *COLOR_BGR2GRAY*, pertenecientes a la librería *OpenCV.* Las imágenes en escala de grises aquí obtenidas deberán ser rectificadas para poder ser usadas en la generación del mapa de profundidades y la detección de personas.

Realizar el procesamiento a partir de imágenes en escala de grises representa una significativa ventaja en tiempo y recursos utilizados, frente al uso directo de imágenes a color. Cada pixel en una imagen en grises está descrito por un único valor en el rango 0-255,

mientras que los pixeles en una imagen a color en formato RGB están descritos por tres canales correspondientes a cada color: rojo, verde, azul. Esta relación de 3 a 1 es un aspecto importante a tomar en cuenta cuando la capacidad de procesamiento es un recurso escaso.

5.5.3. Rectificación

Para poder producir mapas de profundidades, es necesario rectificar las imágenes capturadas utilizando los parámetros obtenidos por el proceso de calibración previamente realizado. Las imágenes correspondientes a la vista izquierda y vista derecha en escala de grises son rectificadas usando la librería para Python: *stereovision*. El resultado es un nuevo par de imágenes: la vista izquierda rectificada y la vista derecha rectificada, las cuales están listas para ser usadas en la generación del mapa de disparidad.

El resultado de aplicar todas estas operaciones de preprocesamiento a la vista izquierda de una imagen estéreo se muestra en la *Figura 23*.

Figura 23. Vista izquierda rectificada.

5.5.4. Mapa de disparidad y profundidad

Para generar el mapa de profundidades de la escena observada por las cámaras de la tarjeta *StereoPi* se utilizan las herramientas facilitadas por la librería *OpenCV.* La función *compute* relacionada a la clase *StereoBM* (del inglés "*Stereo Block Matching"*) permite obtener la matriz de disparidades entre las imágenes rectificadas anteriormente. Al realizar el cálculo de las disparidades, esta función ejecuta un algoritmo de coincidencia de bloques para hallar la correspondencia entre la vista izquierda y derecha de la imagen estéreo, utilizando los parámetros guardados previamente en el archivo *"3dmap_set.txt"*.

Una operación de escalado es aplicada a la matriz de disparidades para obtener valores entre 0 y 255 que podrán ser visualizados como un mapa de profundidades, donde los objetos más cercanos se graficarán con color rojo y los más alejados con azul.

Los valores de disparidad pueden ser transformados en valores de profundidad mediante la *Ecuación 2*.

(Ecuación 2)

$$Z = \text{baseline} * f / (d + \text{doffs})$$

Donde:

- **Z** es la profundidad en cm
- **baseline** o línea base es la distancia horizontal en cm entre las cámaras
- **f** es la distancia focal en pixeles
- **d** es el valor de disparidad
- **doffs** es la diferencia en el eje x de los puntos principales de ambas cámaras

Se calcularon los valores aproximados de dichas constantes de manera experimental. Utilizado los valores encontrados, la transformación a profundidad se realiza mediante la *Ecuación 3*.

(Ecuación 3)

$$Z = 8350 / (d - 65) \quad [\text{cm}]$$

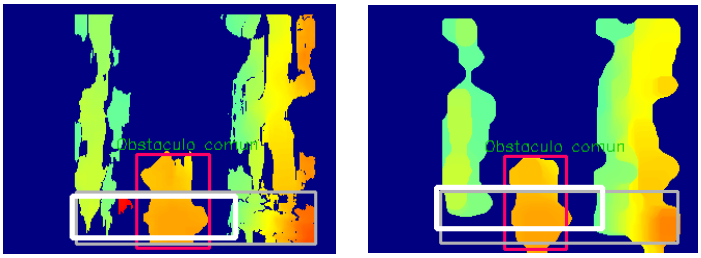
Sin embargo, para disminuir el número de operaciones matriciales, la mayor parte de las tareas de procesamiento utilizarán el mapa de disparidades. Los valores de profundidad se estimarán únicamente para puntos específicos donde esta transformación sea necesaria.

Debido a que este trabajo está orientado a la navegación en entornos de espacio reducido, y para una mejor identificación de los objetos de interés, solamente se consideran los valores de disparidad correspondientes a profundidades entre 0,5 y 1,5 metros. Los valores que se encuentran fuera de este rango son sustituidos por cero y son ignorados en los siguientes pasos de procesamiento.

5.5.5. Morfología matemática

Una vez generado el mapa de profundidades, se realizan varias operaciones de morfología matemática para mejorar los resultados de la búsqueda de la ruta crítica y obstáculos. Se utiliza una operación de apertura con un disco de diámetro 10 como elemento estructural, para eliminar los errores pequeños se puedan generar. Posteriormente, se realizan una operación

de clausura utilizando un disco de 15 píxeles de diámetro para dar solidez a las áreas detectadas como objetos o paredes cercanos.



a) b)

Figura 24. Mapa de profundidades. a) Antes de las operaciones de morfología matemática. b) Después de las operaciones de morfología matemática.

5.5.6. Identificación de la ruta crítica

Se parte de la selección de una porción de la imagen rectificada. Los robots autónomos terrestres se desplazan sobre el suelo, el cual presenta muy pocas variaciones de nivel en espacios cerrados, por lo que analizar la totalidad de la captura en búsqueda de la ruta crítica resulta innecesario y es una posible causa resultados erróneos. Por dicha razón, lo más acertado es utilizar un área parcial de la imagen ubicada en la zona inferior, donde se visualizarán los obstáculos que puedan encontrarse en la ruta del robot. Se ha seleccionado un área que ocupa el 20 por ciento de la altura y la totalidad de la anchura útil del mapa de disparidades. Este espacio de búsqueda es la zona más adecuada para detectar cualquier obstáculo. Existen márgenes laterales vacíos en las imágenes producidos durante la generación del mapa de profundidades debido a los parámetros utilizados.

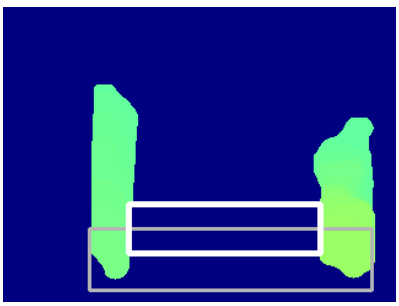


Figura 25. Ruta crítica identificada en el mapa de profundidad.

Mediante una búsqueda exhaustiva con pasos de *s=4* píxeles, se evalúa el espacio horizontal existente entre los segmentos con mayor disparidad de cada fila analizada. Este espacio representa la distancia entre los objetos más cercanos a la cámara, en una altura y profundidad específicas. Se almacena temporalmente la mayor distancia local encontrada en la fila correspondiente, para posteriormente seleccionar la menor de las distancias locales

como ruta crítica. De esta manera se logra identificar la ruta disponible más espaciosa, y conocer la sección más estrecha de la misma, como se señala en la *Figura 25* y *Figura 26*.

Figura 26. Ruta crítica identificada en la vista izquierda.

Adicionalmente, este algoritmo analizará si la ruta crítica encontrada está interrumpida por algún obstáculo mediante la extensión de la misma secuencia hacia los extremos. Este paso le proporcionará al robot de manera sencilla la información sobre la existencia de obstáculos. El tamaño de los obstáculos es registrado y es comparado con un mínimo establecido para evitar la detección de falsos positivos.

Para la selección del umbral de disparidad que permite diferenciar a los objetos cercanos que serán tomados en cuenta durante la medición de distancias horizontales, se ha utilizado un filtro que ignora los valores menos frecuentes en el mapa de disparidades. Esta filtración evita los errores que pueden producirse debido al ruido impulsivo que presenta el algoritmo de correspondencia de bloques, que provoca la asignación de valores de disparidad máxima en zonas pequeñas de la imagen.

5.5.7. Estimación y comparación de anchura

La ruta crítica ya identificada se encuentra medida en número de pixeles. Es necesario transformarla a unidades de distancia mediante la *Ecuación 4*.

(Ecuación 4)

$$\Delta Y = \mathrm{dpix} * Z * k_1 + k_2$$

Donde:

- ΔY es la distancia entre objetos, medida en cm
- dpix es la distancia entre objetos, medida en número de pixeles
- k_1, k_2 son constantes de transformación

La distancia ΔY es proporcional a la profundidad de los objetos que se está analizando y a la constante de transformación k_1. Los valores de las constantes k_1 y k_2 han sido aproximados de manera experimental, resultando en la *Ecuación 5*.

(Ecuación 5)

$$\Delta \mathrm{Y} = \mathrm{dpix} * \mathrm{Z} * 0.0036 - 0.82$$

Esta medida, que correspondiente a la anchura de la ruta en cm, es comparada con la dimensión del ancho del robot previamente informadas por el usuario, multiplicada por un factor de seguridad. El resultado de esta comparación indicará si el espacio es suficiente para que el robot pueda movilizarse a través del mismo, mediante una variable binaria llamada "*espacio suficiente",* cuyo valor es *Verdadero* cuando se satisface la *Ecuación 6*. Esta información será esencial para la selección de las acciones a ejecutar.

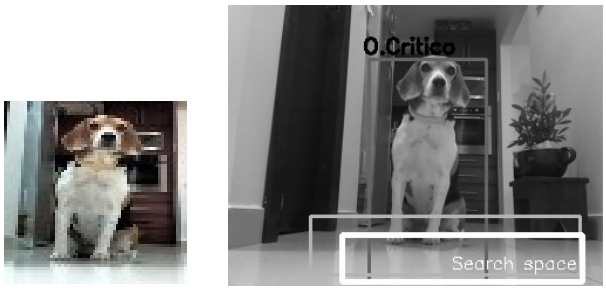
(Ecuación 6)

$$espacio\ suficiente : \Delta \mathrm{Y} > w_{robot} * fs$$

Donde w_{robot} es el ancho del robot en cm, y *fs* es el factor de seguridad mayor a la unidad, cuyo valor en esta implementación es *fs=1.2*.

5.6. Clasificación de obstáculos

Cuando un obstáculo es detectado en la etapa previa, se selecciona una porción rectangular de la captura correspondiente a la ubicación del obstáculo detectado para ser clasificada por el modelo entrenado anteriormente. Este segmento debe ser preprocesado de la misma manera que las imágenes de entrenamiento para coincidir con el tipo de datos utilizados como entrada de la red. Por esta razón, se realiza el redimensionamiento de éste a 64x64 píxeles, el realce de contraste mediante la ecualización de histograma, y normalización como se detalló en la sección *5.4.2. Preprocesamiento de datos*. Adicionalmente, el tamaño de entrada del modelo hace que sea necesario agregar una cuarta dimensión. Esta última operación es realizada por la función *blobFromImage* perteneciente al paquete especializado en redes neuronales profundas "*dnn"* de la librería *OpenCV*, obteniendo como resultado una imagen de dimensiones 1x3x64x64 lista para ser procesada por el clasificador.



a) b)

Figura 27. Clasificación de obstáculos. a) *Blob* analizado.
b) Vista izquierda etiquetada con la categoría asignada.

Dicha imagen es categorizada por la red neuronal convolucional ya entrenada, con el fin de clasificar los objetos en dos posibles tipos: común, o crítico. La salida obtenida es un vector que contiene dos valores, que corresponden a la probabilidad estimada de que la imagen analizada pertenezca a cada una de las clases posibles. Al identificar el índice del mayor valor entre los dos obtenidos, se conoce la categoría resultante. En la ventana de visualización de video se dibuja un recuadro alrededor del obstáculo detectado y la etiqueta “obstáculo común” u “obstáculo crítico”, según la categoría asignada, como se muestra en la *Figura 27*.

5.7. Módulo de decisiones

El mapa de profundidades le permite al robot conocer si cuenta con el espacio suficiente para movilizarse hacia su destino por la ruta actual. Además, puede detectar la existencia de obstáculos y calcular si un obstáculo puede ser evadido fácilmente o representa un impedimento para continuar su navegación. La clasificación de imágenes permite identificar si los obstáculos detectados son del tipo comunes o críticos, lo que determinará la decisión que se tomará.

En los casos en que el obstáculo detectado en la vía no sea un objeto crítico, el robot podrá manipularlo para continuar su navegación sin mayores cambios de ruta. Por otro lado, en los casos en que el obstáculo es una persona o mascota, el robot tomará acciones alternativas como detenerse y emitir una alarma sonora hasta que ésta le permita el paso, o caso contrario buscar una ruta distinta. El proceso de toma de decisiones se detalla en la *Figura 28*.

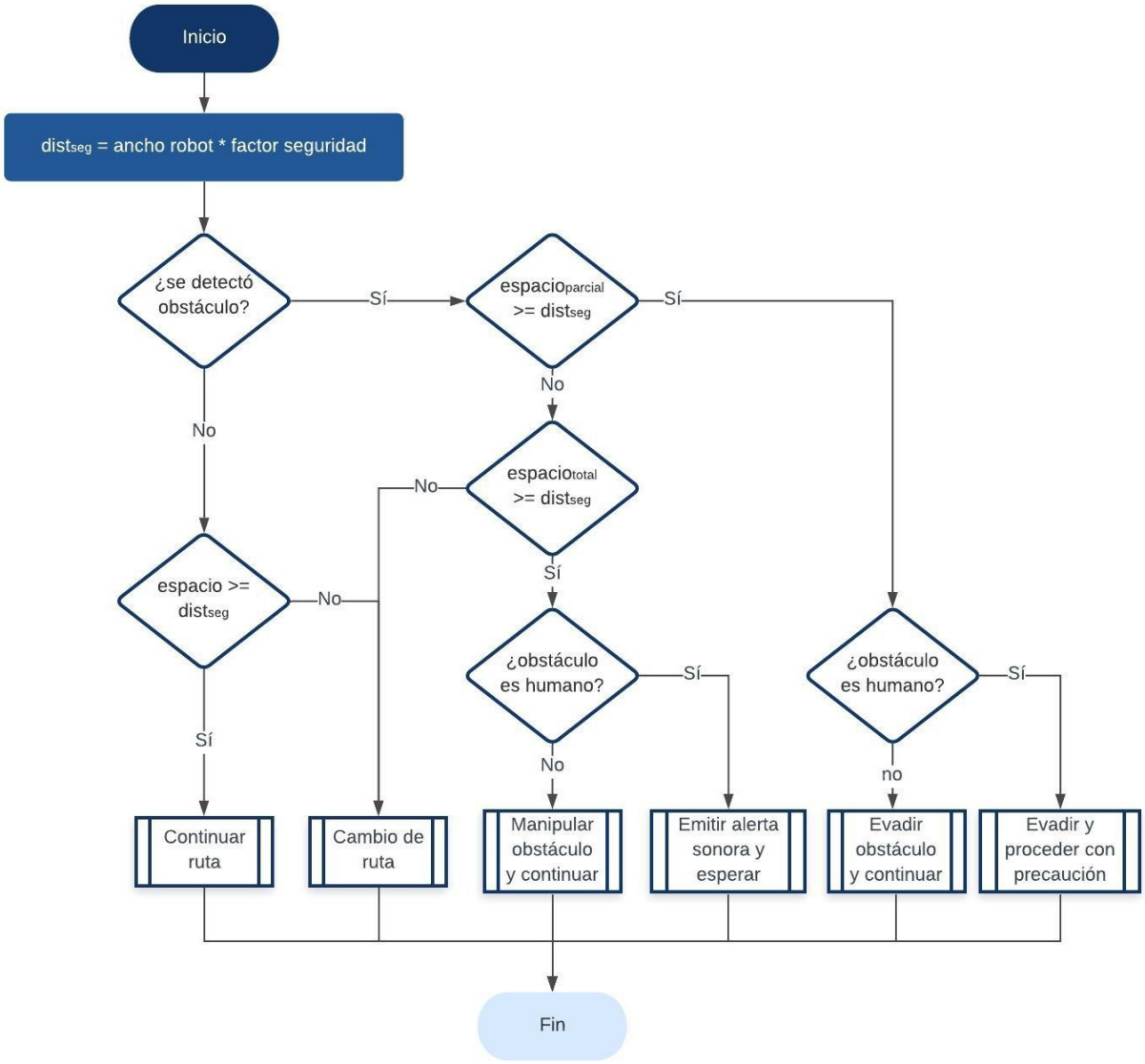


Figura 28. Flujograma de la etapa de decisiones.

6. Pruebas y resultados

Se realizaron pruebas en 9 escenarios distintos, correspondientes a zonas comunes del hogar, capturados bajo distintas circunstancias: sin obstáculo, con la presencia de un obstáculo común, y con la presencia de un obstáculo crítico. Los objetos utilizados como obstáculos son artículos domésticos o de oficina que podrían encontrarse casualmente sobre el suelo como pelotas, juguetes, calzado, esferográficos, etc. En representación de los obstáculos críticos correspondientes a mascotas y personas (principalmente niños), algunas capturas incluyen un perro y dos juguetes: un peluche de gato, y un muñeco de bebé.

La captura de las imágenes fue ejecutada por la misma *StereoPi* que se utilizó durante el desarrollo de este trabajo, posicionada de forma que los lentes de las cámaras se encuentren a 7,5 cm de altura y su línea de visión central forme un ángulo de elevación de 15 grados con respecto al suelo. Esta configuración pretende simular la ubicación que las cámaras tendrían en un robot móvil, y permite optimizar el rango de visión para detectar desde objetos pequeños hasta objetos de gran tamaño, ya sea que se encuentren cerca o lejos de la *StereoPi*. Los resultados arrojados por el software desarrollado, se detallan a continuación para cada una de las escenas.

6.1. Escenarios de prueba

Los escenarios utilizados para evaluar los resultados son caminos de hasta un metro de ancho, en los cuales la herramienta desarrollada tendría mayor relevancia. Estas zonas estrechas corresponden a pasillos, puertas y espacios entre muebles de distintas habitaciones de una casa, ubicadas a distancias entre 50 cm y 150 cm delante de las cámaras, en concordancia con el rango de funcionamiento del software. El parámetro de anchura ingresado en el software fue de 20 cm, lo cual determina si el espacio disponible encontrado es considerado suficiente para que el robot pueda movilizarse.

6.1.1. Escena B

La primera escena, en contraste con todas las posteriores, corresponde a un espacio amplio, sin paredes ni objetos cercanos, por lo que todo lo que se ve en el fondo es ignorado por el algoritmo. El espacio disponible es máximo, y solamente los obstáculos de prueba son considerados en el mapa de profundidades. Como obstáculo común se utilizó una pelota y como obstáculos críticos un muñeco de bebé y un peluche de gato, como se observa en la *Figura 29*.

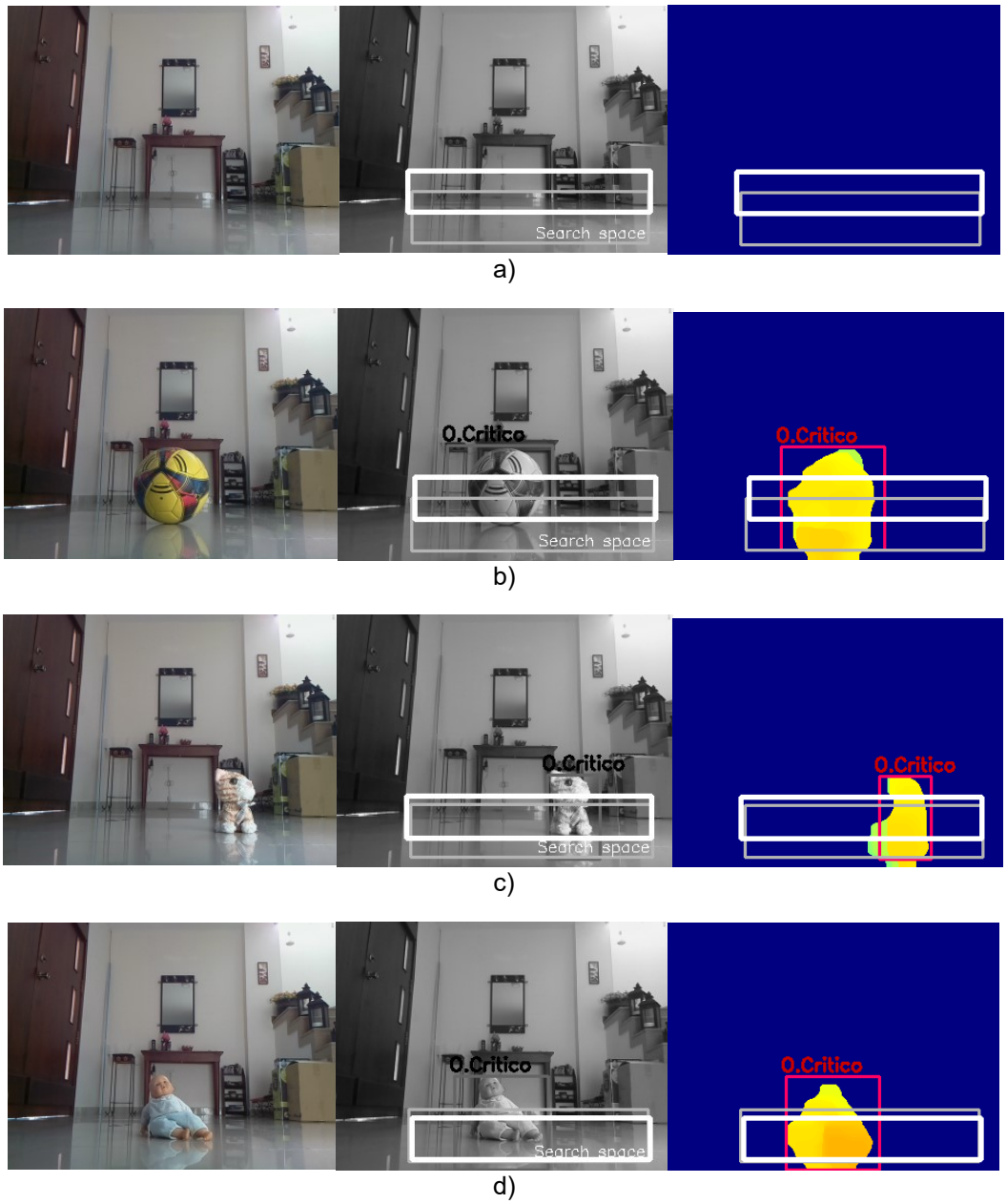


Figura 29. Escena B. a) sin obstáculo. b) con obstáculo común (pelota). c) con obstáculo crítico (gato). d) con obstáculo crítico (bebé).

Los resultados obtenidos a partir de estas capturas son:

Tabla 7. Resultados obtenidos de la escena B

Caso	*Distancia total [cm]*	*Existe obstáculo [Si / No]*	*Distancia parcial [cm]*	*Tipo de obstáculo [Común / Crítico]*	*Decisión resultante*
a)	166	No	-	-	Espacio suficiente: Continuar ruta
b)	66	Si	30	1: Crítico	Evadir y proceder con precaución
c)	68	Si	41	1: Crítico	Evadir y proceder con precaución
d)	59	Si	27	1: Crítico	Evadir y proceder con precaución

6.1.2. Escena C

La escena C corresponde al espacio entre muebles de la sala del hogar de prueba. La distancia total de separación entre los muebles es de 50 cm. Como obstáculos comunes se utilizaron un rollo de cinta adhesiva y un marcador; como obstáculos críticos aparecen un gato y un perro, como se observa en la *Figura 30*.

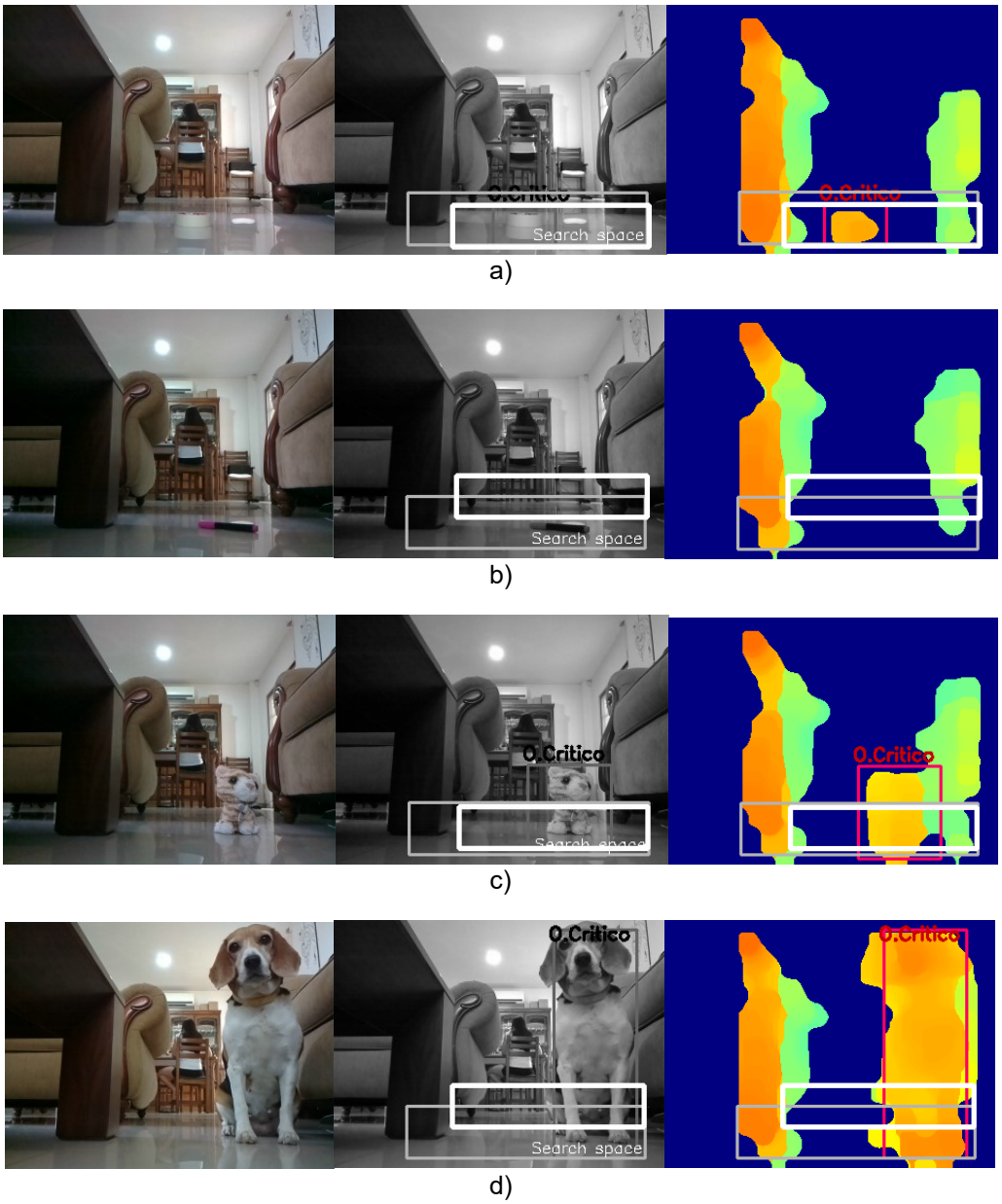


Figura 30. Escena C. a) sin obstáculo. b) con obstáculo común (marcador). c) con obstáculo crítico (gato). d) con obstáculo crítico (perro).

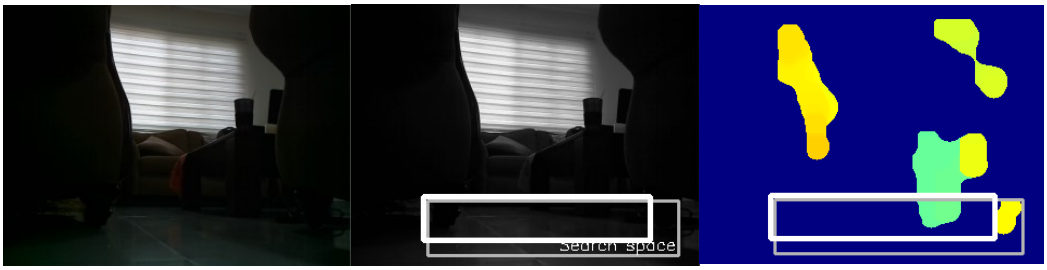
Los resultados obtenidos a partir de estas capturas son:

Tabla 8. Resultados obtenidos de la escena C

Caso	*Distancia total [cm]*	*Existe obstáculo [Si / No]*	*Distancia parcial [cm]*	*Tipo de obstáculo [Común / Crítico]*	*Decisión resultante*
a)	43	Si	22	1: Crítico	Espacio insuficiente por obstáculo: Emitir alerta sonora y esperar
b)	42	No	-	-	Espacio suficiente: Continuar ruta
c)	42	Si	17	1: Crítico	Espacio insuficiente por obstáculo: Emitir alerta sonora y esperar
d)	43	Si	24	1: Crítico	Espacio insuficiente por obstáculo: Emitir alerta sonora y esperar

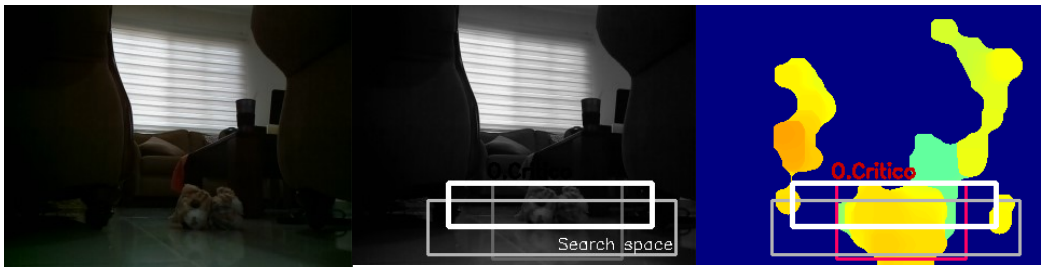
6.1.3. Escena D

La escena D corresponde al espacio entre muebles de la sala del hogar de prueba. La distancia total de separación entre los muebles es de 45 cm. Como obstáculo común se utilizó un par de auriculares, y como obstáculo crítico un gato, como se observa en la *Figura 31*.



a)

b)



c)

Figura 31. Escena D. a) sin obstáculo. b) con obstáculo común (audífonos). c) con obstáculo crítico (gato).

Los resultados obtenidos a partir de estas capturas son:

Tabla 9. Resultados obtenidos de la escena D

Caso	*Distancia total [cm]*	*Existe obstáculo [Si / No]*	*Distancia parcial [cm]*	*Tipo de obstáculo [Común / Crítico]*	*Decisión resultante*
a)	64	No	-	-	Espacio suficiente: Continuar ruta
b)	55	Si	22	0: Común	Manipular obstáculo para poder continuar
c)	54	Si	12	1: Crítico	Espacio insuficiente por obstáculo: Emitir alerta sonora y esperar

6.1.4. Escena E

La escena E corresponde a un pasillo ubicado junto al comedor del hogar de prueba. La distancia total disponible es de 70 cm. Como obstáculo común se utilizó un juguete, y como obstáculo crítico un perro, como se observa en la *Figura 32*.

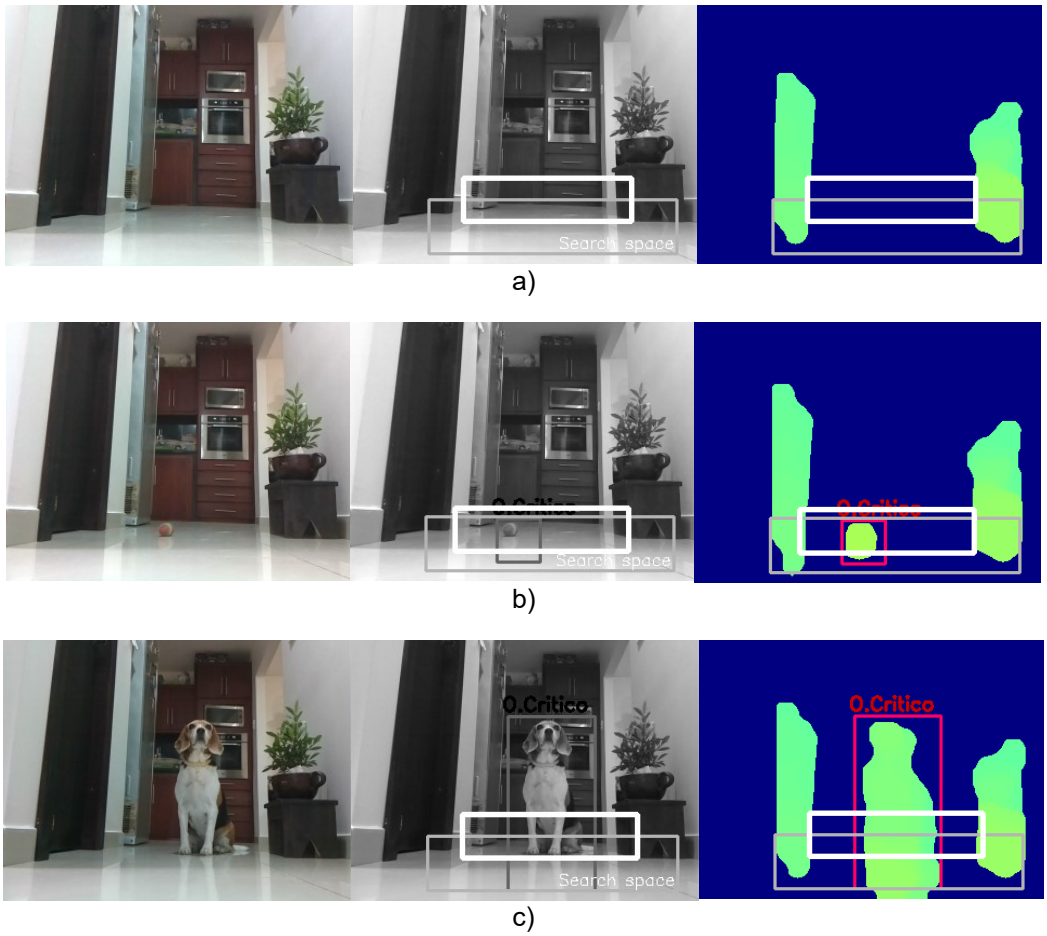


Figura 32. Escena E. a) sin obstáculo. b) con obstáculo común (juguete). c) con obstáculo crítico (perro).

Los resultados obtenidos a partir de estas capturas son:

Tabla 10. Resultados obtenidos de la escena E

Caso	*Distancia total [cm]*	*Existe obstáculo [Si / No]*	*Distancia parcial [cm]*	*Tipo de obstáculo [Común / Crítico]*	*Decisión resultante*
a)	65	No	-	-	Espacio suficiente: Continuar ruta
b)	64	Si	36	1: Crítico	Evadir y proceder con precaución
c)	68	Si	20	1: Crítico	Espacio insuficiente por obstáculo: Emitir alerta sonora y esperar

6.1.5. Escena F

La escena F corresponde al espacio disponible entre la mesa de comedor y el aparador en el hogar de prueba. La distancia total disponible es de 55 cm. Como obstáculo común se utilizó un frasco de medicina, y como obstáculo crítico un gato en dos posiciones y ubicaciones distintas, como se observa en la *Figura 33*.

a)

b)

c)

d)

Figura 33. Escena F. a) sin obstáculo. b) con obstáculo común (frasco de medicina). c) con obstáculo crítico (gato - lateral). d) con obstáculo crítico (gato - frontal).

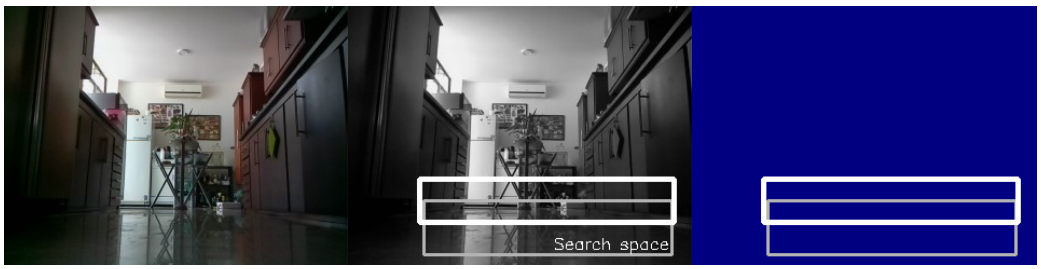
Los resultados obtenidos a partir de estas capturas son:

Tabla 11. Resultados obtenidos de la escena F

Caso	*Distancia total [cm]*	*Existe obstáculo [Si / No]*	*Distancia parcial [cm]*	*Tipo de obstáculo [Común / Crítico]*	*Decisión resultante*
a)	51	No	-	-	Espacio suficiente: Continuar ruta
b)	41	Si	18	0: Común	Manipular obstáculo para poder continuar
c)	40	Si	18	1: Crítico	Espacio insuficiente por obstáculo: Emitir alerta sonora y esperar
d)	48	Si	19	1: Crítico	Espacio insuficiente por obstáculo: Emitir alerta sonora y esperar

6.1.6. Escena G

La escena G corresponde al pasillo de la cocina del hogar de prueba. La distancia total disponible es de 118 cm. Como obstáculo común se utilizó un plato de mascota, y como obstáculo crítico un perro en distintas posiciones, como se observa en la *Figura 34*.



a)

b)

c)

d)

Figura 34. Escena G. a) sin obstáculo. b) con obstáculo común (plato de mascota). c) con obstáculo crítico (perro - frontal). d) con obstáculo crítico (perro - posterior).

Los resultados obtenidos a partir de estas capturas son:

Tabla 12. Resultados obtenidos de la escena G

Caso	*Distancia total [cm]*	*Existe obstáculo [Si / No]*	*Distancia parcial [cm]*	*Tipo de obstáculo [Común / Crítico]*	*Decisión resultante*
a)	166	No	-	-	Espacio suficiente: Continuar ruta
b)	65	Si	32	0: Común	Evadir obstáculo y continuar
c)	86	Si	36	1: Crítico	Evadir y proceder con precaución
d)	88	Si	48	1: Crítico	Evadir y proceder con precaución

6.1.7. Escena H

La escena H corresponde al espacio disponible entre el pie de cama y la pared de la habitación principal del hogar de prueba. La distancia total disponible es de 75 cm. Como obstáculo común se utilizó una pieza da calzado femenino, y como obstáculo crítico un perro, como se observa en la *Figura 35*.

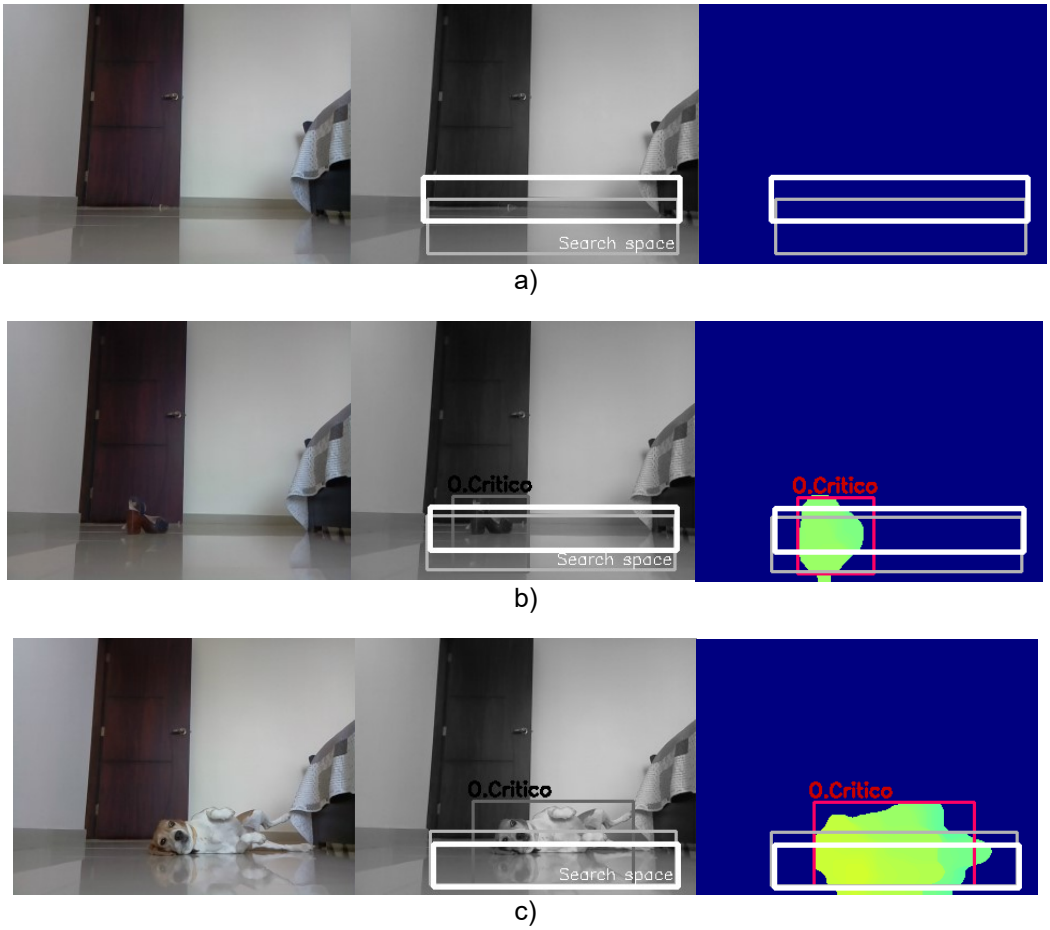


Figura 35. Escena H. a) sin obstáculo. b) con obstáculo común (calzado femenino). c) con obstáculo crítico (perro).

Los resultados obtenidos a partir de estas capturas son:

Tabla 13. Resultados obtenidos de la escena H

Caso	*Distancia total [cm]*	*Existe obstáculo [Si / No]*	*Distancia parcial [cm]*	*Tipo de obstáculo [Común / Crítico]*	*Decisión resultante*
a)	166	No	-	-	Espacio suficiente: Continuar ruta
b)	99	Si	63	1: Crítico	Evadir y proceder con precaución
c)	81	Si	24	1: Crítico	Espacio insuficiente por obstáculo: Emitir alerta sonora y esperar

6.1.8. Escena I

La escena I corresponde a la puerta de uno de los cuartos del hogar de prueba. La distancia total disponible es de 70 cm. Como obstáculo común se utilizó un coche de juguete, y como obstáculo crítico un gato, como se observa en la *Figura 36*.

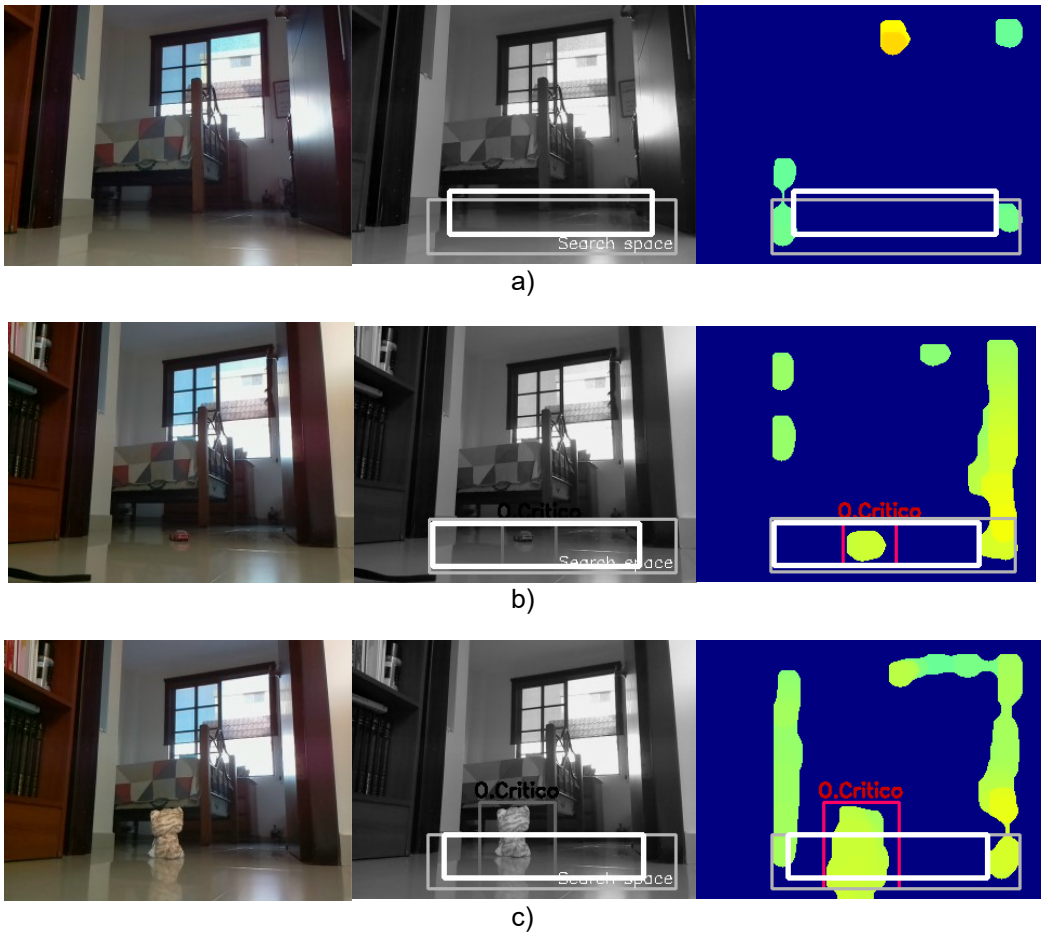


Figura 36. Escena I. a) sin obstáculo. b) con obstáculo común (juguete). c) con obstáculo crítico (gato).

Los resultados obtenidos a partir de estas capturas son:

Tabla 14. Resultados obtenidos de la escena I

Caso	*Distancia total [cm]*	*Existe obstáculo [Si / No]*	*Distancia parcial [cm]*	*Tipo de obstáculo [Común / Crítico]*	*Decisión resultante*
a)	91	No	-	-	Espacio suficiente: Continuar ruta
b)	66	Si	29	1: Crítico	Evadir y proceder con precaución
c)	64	Si	31	1: Crítico	Evadir y proceder con precaución

6.1.9. Escena J

La escena J corresponde al espacio disponible entre la cama y la pared de un cuarto femenino del hogar de prueba. La distancia total disponible es de 100 cm. Como obstáculo común se utilizaron unas sandalias, y como obstáculo crítico un muñeco de bebé, como se observa en la *Figura 37*.

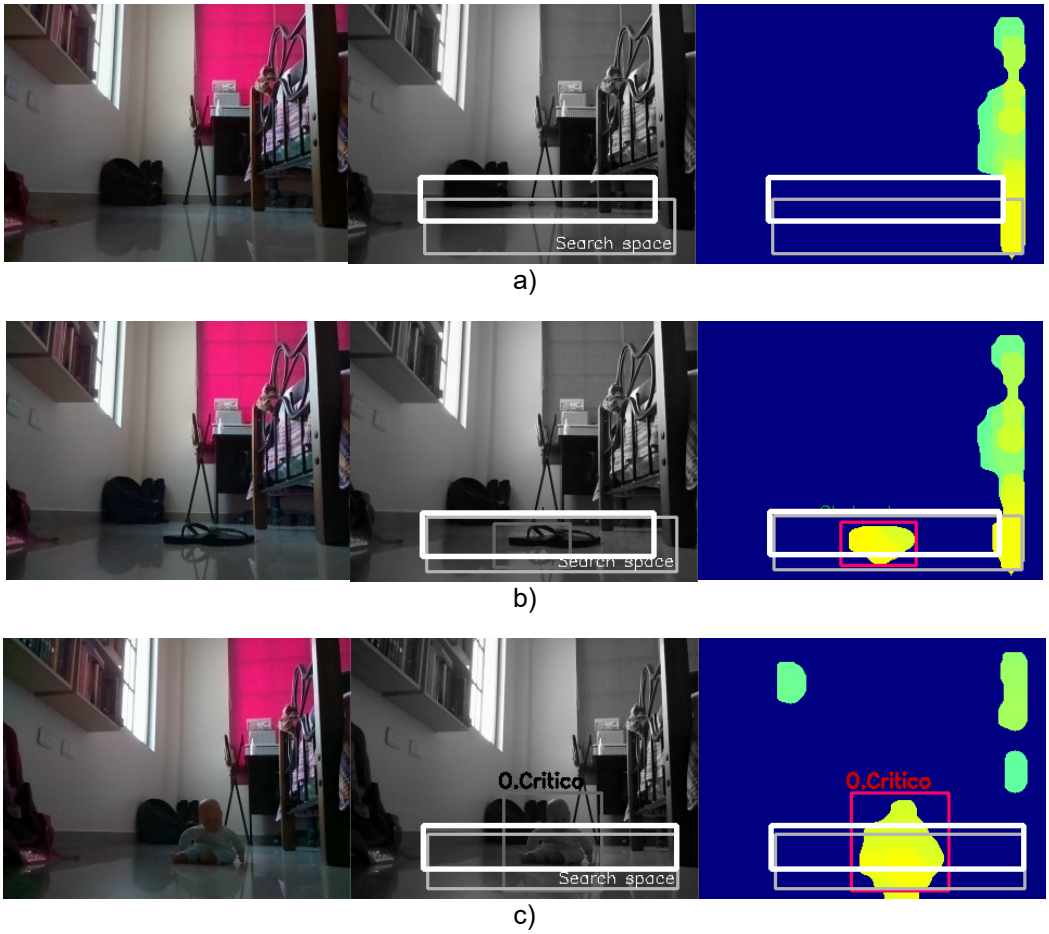


Figura 37. Escena J. a) sin obstáculo. b) con obstáculo común (sandalias). c) con obstáculo crítico (bebé).

Los resultados obtenidos a partir de estas capturas son:

Tabla 15. Resultados obtenidos de la escena J

Caso	*Distancia total [cm]*	*Existe obstáculo [Si / No]*	*Distancia parcial [cm]*	*Tipo de obstáculo [Común / Crítico]*	*Decisión resultante*
a)	69	No	-	-	Espacio suficiente: Continuar ruta
b)	66	Si	22	0: Común	Manipular obstáculo para poder continuar
c)	76	Si	26	1: Crítico	Evadir y proceder con precaución

6.2. Recopilación de resultados

Los resultados obtenidos en las 31 capturas previamente detalladas se han recopilado para ser evaluados por las métricas que los resuman adecuadamente, de acuerdo a las principales etapas del software desarrollado. Entre las métricas de evaluación utilizadas se encuentran la matriz de confusión, exactitud, precisión, sensibilidad, y *F1*.

La matriz de confusión es una tabla que detalla el número de predicciones correctas e incorrectas por cada clase o categoría. En las filas se ubican las clases predichas, mientras que en columnas se ubican las clases reales de los ejemplos utilizados. A partir de esta matriz se pueden obtener el número de verdaderos positivos, verdaderos negativos, falsos positivos, y falsos negativos, como se indica en la *Tabla 16*.

Tabla 16. Matriz de confusión para la clasificación de dos categorías

Predicciones:	***Categoría real: Negativo / Clase 0***	***Categoría real: Positivo / Clase 1***	***Total***
Negativo / Categoría 0	Verdaderos negativos (VN)	Falsos negativos (FN)	Total de predicciones negativas (VN+FN)
Positivo / Categoría 1	Falsos positivos (FP)	Verdaderos positivos (VP)	Total de predicciones positivas (VP+FP)
Total	Total de ejemplos negativos (VN+FP)	Total de ejemplos positivos (VP+FN)	Total de ejemplos (VP+VN+FP+FN)

La exactitud, también conocida como *accuracy*, representa la proporción entre las predicciones correctas y el total de predicciones. Se formula mediante la *Ecuación 7*.

(Ecuación 7)

$$\boldsymbol{Exactitud} = \frac{VP + VN}{VP + VN + FP + FN}$$

La precisión representa la fracción de predicciones positivas que han sido clasificadas correctamente, y se formula mediante la *Ecuación 8*.

(Ecuación 8)

$$\boldsymbol{Precision} = \frac{VP}{VP + FP}$$

La sensibilidad, también conocida como *recall*, indica la fracción de ejemplos positivos que han sido clasificados correctamente. Se formula mediante la *Ecuación 9*.

(Ecuación 9)

$$\boldsymbol{Sensibilidad} = \frac{VP}{VP + FN}$$

La medida F1, también conocida como *F-meaure*, indica la media armónica entre precisión y sensibilidad, con el fin de representar mediante una única métrica el rendimiento de un algoritmo de clasificación, y está dada por la *Ecuación 10.*

(Ecuación 10)

$$\boldsymbol{F1} = \frac{2 * precision * recall}{precision + recall}$$

6.2.1. Exactitud en la ruta crítica

Los resultados obtenidos por la etapa de identificación de la ruta crítica y estimación de distancias se detallan en la *Tabla 16*. La distancia estimada promedio de cada escena corresponde a la media aritmética de las distancias obtenidas por los distintos casos de una misma escena.

Tabla 17. Error en la estimación de ruta crítica

Escena de prueba	*Distancia estimada promedio: de [cm]*	*Distancia real: dr [cm]*	*Error: e = de - dr [cm]*	*Error porcentual absoluto [%]*
A	89.8	máxima	-	-
C	42.5	50	-7.5	15.0%
D	57.7	45	12.7	28.1%
E	65.7	70	-4.3	6.2%
F	45.0	55	-10.0	18.2%
G	101.3	118	-16.8	14.2%
H	115.3	75	40.3	53.8%
I	73.7	70	3.7	5.2%
J	70.3	100	-29.7	29.7%
			Error promedio	21.3%
			Exactitud	**78.7%**

El error promedio calculado es del 21%. Este error se debe principalmente a que el rango de visibilidad de las cámaras y de generación del mapa de profundidades no alcanza a percibir los extremos de la ruta crítica en algunas de las escenas. Por esta razón, la exactitud resultante en esta etapa es de 79%.

6.2.2. Detección de obstáculos

La evaluación de los resultados obtenidos durante la etapa de detección de obstáculos está caracterizada por la matriz de confusión detallada en la *Tabla 18*.

Tabla 18. Matriz de confusión de la detección de obstáculos

Detectado:	*Real: Sin obstáculos*	*Real: Con obstáculos*	*Total*
Sin obstáculos	8	1	9
Con obstáculos	0	22	22
Total	8	23	31

Donde:

- Verdaderos Positivos (VP): 22
- Verdaderos Negativos (VN): 8
- Falsos Positivos (FP): 0
- Falsos Negativos (FN): 1

$$\boldsymbol{Exactitud} = \frac{22 + 8}{22 + 8 + 0 + 1} = 0.9677 = \mathbf{96.8}\%$$

$$\boldsymbol{Precision} = \frac{22}{22 + 0} = 1 = \mathbf{100}\%$$

Solamente uno de los 31 casos evaluados fue clasificado de manera incorrecta, logrando una exactitud mayor al 95%. Este ejemplo corresponde a un obstáculo que no fue percibido por el algoritmo de detección debido a su pequeño tamaño (*Figura 30.b*).

No se produjo ningún falso positivo; es decir, que todas las escenas sin obstáculos fueron correctamente identificadas, alcanzando una precisión del 100%. Finalmente, se puede sintetizar la evaluación de la detección de obstáculos mediante una medida *F1* = 0.98.

$$\boldsymbol{F1} = \frac{2 * precision * recall}{precision + recall} = \frac{2 * 1 * 0.957}{1 + 0.957} = \mathbf{0.98}$$

6.2.3. Clasificación de obstáculos

En la *Tabla 19* se detallan los resultados obtenidos por la clasificación de obstáculos realizada por la red neuronal convolucional entrenada en este trabajo. Se han tomado en cuenta, para esta evaluación, solamente los 22 casos en los que se detectaron obstáculos.

Tabla 19. Matriz de confusión de la clasificación de obstáculos

Categoría predicha:	*Categoría real: Común*	*Categoría real: Crítico*	*Total*
Común	4	0	4
Crítico	5	13	18
Total	9	13	22

Donde:

- Verdaderos Positivos (VP): 13
- Verdaderos Negativos (VN): 4
- Falsos Positivos (FP): 5
- Falsos Negativos (FN): 0

$$\boldsymbol{Exactitud} = \frac{13+4}{13+4+5+0} = \frac{17}{23} = 0.7391 = \mathbf{73.9}\%$$

$$\boldsymbol{Sensibilidad} = \frac{22}{22+0} = 1 = \mathbf{100}\%$$

Se puede observar que 5 de los 9 obstáculos comunes fueron incorrectamente clasificados como críticos. Esto representa un gran margen de error que disminuye la exactitud a un 74%. Sin embargo; todos los obstáculos críticos fueron acertadamente identificados como tal, lo cual se refleja en una sensibilidad del 100%. De esta manera, el algoritmo de clasificación se caracteriza como conservador, lo cual resulta adecuado para la finalidad del software de salvaguardar la seguridad física de los posibles obstáculos críticos.

$$\boldsymbol{F1} = \frac{2 * precision * recall}{precision + recall} = \frac{2 * 0.8148 * 1}{0.8148 + 1} = \mathbf{0.90}$$

El rendimiento de esta etapa se resume mediante la métrica *F1*=0.90.

6.2.4. Decisiones finales

Finalmente, las decisiones resultantes en la última etapa del software desarrollado se detallan en la *Tabla 20*. Se han clasificado los errores producidos en dos tipos. Los errores poco relevantes son los que modificarían en mínimo grado el comportamiento del robot, mientras que los errores relevantes representan efectos evidentes y no deseados.

Tabla 20. Decisiones finales resultantes

Escenario de prueba		*Decisión calculada*	*Decisión óptima de referencia*
A	*a*	Espacio suficiente: Continuar ruta	Espacio suficiente: Continuar ruta
	b	Evadir y proceder con precaución	Evadir obstáculo y continuar
	c	Evadir y proceder con precaución	Evadir y proceder con precaución
	d	Evadir y proceder con precaución	Evadir y proceder con precaución
C	*a*	Emitir alerta sonora y esperar	Evadir obstáculo y continuar
	b	Espacio suficiente: Continuar ruta	Evadir obstáculo y continuar
	c	Emitir alerta sonora y esperar	Emitir alerta sonora y esperar
	d	Emitir alerta sonora y esperar	Emitir alerta sonora y esperar
D	*a*	Espacio suficiente: Continuar ruta	Espacio suficiente: Continuar ruta
	b	Manipular obstáculo para poder continuar	Manipular obstáculo para poder continuar
	c	Emitir alerta sonora y esperar	Emitir alerta sonora y esperar
E	*a*	Espacio suficiente: Continuar ruta	Espacio suficiente: Continuar ruta
	b	Evadir y proceder con precaución	Evadir obstáculo y continuar
	c	Emitir alerta sonora y esperar	Emitir alerta sonora y esperar
F	*a*	Espacio suficiente: Continuar ruta	Espacio suficiente: Continuar ruta
	b	Manipular obstáculo para poder continuar	Manipular obstáculo para poder continuar
	c	Emitir alerta sonora y esperar	Emitir alerta sonora y esperar
	d	Emitir alerta sonora y esperar	Emitir alerta sonora y esperar
G	*a*	Espacio suficiente: Continuar ruta	Espacio suficiente: Continuar ruta
	b	Evadir obstáculo y continuar	Evadir obstáculo y continuar
	c	Evadir y proceder con precaución	Evadir y proceder con precaución
	d	Evadir y proceder con precaución	Evadir y proceder con precaución
H	*a*	Espacio suficiente: Continuar ruta	Espacio suficiente: Continuar ruta
	b	Evadir y proceder con precaución	Evadir obstáculo y continuar
	c	Emitir alerta sonora y esperar	Evadir y proceder con precaución
I	*a*	Espacio suficiente: Continuar ruta	Espacio suficiente: Continuar ruta
	b	Evadir y proceder con precaución	Evadir obstáculo y continuar
	c	Evadir y proceder con precaución	Evadir y proceder con precaución
J	*a*	Espacio suficiente: Continuar ruta	Espacio suficiente: Continuar ruta
	b	Manipular obstáculo para poder continuar	Espacio suficiente: Continuar ruta
	c	Evadir y proceder con precaución	Evadir y proceder con precaución
		Errores poco relevantes	***5***
		Errores relevantes	***3***
		Total errores	***8***

El número total de casos evaluados es 31, por lo tanto, el porcentaje de error en las decisiones resultantes se calculan dividiendo el número de errores encontrados para dicho valor. De esta manera, el porcentaje de errores poco relevantes es el 16%, mientras que los errores relevantes representan solamente el 10%. La totalidad de errores obtenidos suman un 26% de los casos de prueba, y se producen por la acumulación de los errores durante todas las etapas previas.

6.2.5. Velocidad de procesamiento en tiempo real

El tiempo empleado para el procesamiento completo varía entre 0.5 y 0.6 segundos por captura, dependiendo de la cantidad de objetos observados en el mapa de profundidades y de la presencia o ausencia de obstáculos. De esta manera, la velocidad promedio de ejecución es de 1.8 FPS (cuadros por segundo).

El principal factor limitante de la velocidad de procesamiento es la capacidad de la unidad central de procesamiento CPU de la tarjeta *StereoPi*. Al utilizar redes neuronales convolucionales, las condiciones óptimas para maximizar la velocidad incluyen la utilización de unidades de procesamiento gráfico GPU.

7. Conclusiones y trabajo futuro

7.1. Conclusiones

Se desarrolló una herramienta de software que, a partir de las imágenes capturadas por una cámara estéreo, es capaz de aproximar el ancho de la ruta que se encuentra por delante, identificar la presencia de obstáculos y clasificarlos según su naturaleza, con el fin de tomar la decisión más conveniente para un robot de navegación autónoma sin poner en peligro la integridad de niños o mascotas que se encuentren en el entorno.

Se implementó un algoritmo de detección de obstáculos mediante la búsqueda exhaustiva en la parte inferior del mapa de profundidades. Este algoritmo encuentra la ruta crítica más cercana y los objetos presentes en ella, en el rango de profundidad entre 0,5m y 2m. Se estima la distancia parcial y total de la ruta crítica en centímetros, en base a su profundidad. El rendimiento de la detección está caracterizado por la medida *F1=0.98*, y una precisión del 100%, demostrando alta robustez en la localización de obstáculos de tamaño considerable.

Se entrenó un modelo de clasificación mediante aprendizaje profundo, que asigna las capturas de los obstáculos previamente detectados a una de las dos posibles categorías: obstáculos comunes y obstáculos críticos. La arquitectura de la red neuronal incluye 4 capas convolucionales y 1 capa densa oculta. Los resultados obtenidos son conservadores, evitando la generación de falsos negativos, y están representados por la métrica *F1=0.90*, sensibilidad de 100% y exactitud de 74%.

Una etapa final procesa la información arrojada por los procedimientos previos y arroja la decisión más acertada de acuerdo a estos datos. Se han considerado seis distintas decisiones posibles: continuar ruta, evadir obstáculo y continuar, evadir y proceder con precaución, manipular obstáculo para poder continuar, emitir una alerta sonora y esperar que el objeto crítico permita el paso, o cambiar la ruta. De entre las decisiones calculadas durante las pruebas realizadas, se obtuvo que en menos del 10% de decisiones el comportamiento predicho difiere de manera significativa con las acciones óptimas a ejecutar.

El software desarrollado fue implementado en una tarjeta *StereoPi* para la evaluación de los resultados obtenidos en 31 casos de prueba distribuidos en 9 distintos escenarios de una casa familiar. La velocidad de procesamiento en el hardware utilizado alcanza un promedio de 2 FPS. A pesar de que el rendimiento de esta herramienta está lejos de ser óptimo, presenta una buena alternativa para aplicaciones de investigación en robótica que utilicen únicamente una tarjeta basada en *RaspberryPi* para el procesamiento de información.

7.2. Líneas de trabajo futuro

Para mejorar los resultados del software desarrollado en la estimación de distancias y detección de obstáculos, se considera importante ampliar el ángulo de visibilidad de las cámaras. Esto se puede lograr a través de cámaras con mayor campo de visión; o utilizando cámaras adicionales instaladas con orientaciones distintas.

Las pruebas de campo en un robot real se encuentran fuera del alcance del trabajo presentado. Sin embargo; llevar a cabo esta etapa adicional de pruebas es necesario para observar de manera integral el funcionamiento de este software ante situaciones reales. Con este fin, se recomienda su implementación sobre un agente autónomo doméstico de características similares a los robots aspiradores disponibles en el mercado actual.

A su vez, se podría personalizar el modelo de clasificación para diferenciar objetos de interés específicos o en categorías distintas a las aquí mencionadas, para entornos y aplicaciones especializadas, o para la industria. Una alternativa para mejorar los resultados de la clasificación de obstáculos presentada es la elaboración de una base de datos adaptada específicamente para ambientes y artículos que se encuentren en medios domésticos como casas, departamentos, u oficinas.

Con el fin de optimizar la velocidad de procesamiento y permitir el uso de modelos más complejos, se considera adecuado emplear las tarjetas embebidas como la *StereoPi* exclusivamente para la captura de imágenes y operaciones muy sencillas de preprocesamiento. Esta información debería ser enviada a un ordenador con disponibilidad de una unidad de procesamiento gráfico GPU que sea capaz de ejecutar eficientemente dichos algoritmos.

8. Bibliografía

Alom, Z., Taha, T. M., Yakopcic, C., Westberg, S., Nasrin, S., & Asari, V. K. (2017). The History Began from AlexNet: Comprehensive Survey on Deep Learning Approaches. Recuperado a partir de https://arxiv.org/pdf/1803.01164.pdf

ams AG. (2020). Stereo Vision. Recuperado 20 de julio de 2020, a partir de https://ams.com/stereovision

Aslam, A., & Ansari, M. S. (2019). Depth-Map Generation using Pixel Matching in Stereoscopic Pair of Images. Recuperado a partir de http://arxiv.org/abs/1902.03471

Ayache, N. (1991). Recovering the third dimension. En P. T. Sander (Ed.), *Artificial Vision for Mobile Robots: Stereo Vision and Multisensory Perception* (pp. 7-8). Cambridge, Mass.: The Massachusetts Institute of Technology Press. Recuperado a partir de https://books.google.com.ec/books?hl=es&lr=&id=qkWrml_sTrQC&oi=fnd&pg=PP19&dq=stereo+vision&ots=KoGB6ml81J&sig=7AilyQl67CFNvD1v2EoNtOiy1KM&redir_esc=y#v=onepage&q=stereo vision&f=false

Bloesch, M., Czarnowski, J., Clark, R., Leutenegger, S., & Davison, A. J. (2018). CodeSLAM - Learning a Compact, Optimisable Representation for Dense Visual SLAM. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2560-2568. Recuperado a partir de https://doi.org/10.1109/CVPR.2018.00271

Cooper, G. (2019). New Vision Technologies For Real-World Applications. Recuperado 27 de julio de 2020, a partir de https://semiengineering.com/new-vision-technologies-for-real-world-applications/

Dadouche, A. (2018). Machine Learning in a Box (Part 2): Project Methodologies. Recuperado 31 de mayo de 2020, a partir de https://dzone.com/articles/machine-learning-in-a-box-week-2-project-methodolo-1

Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. En *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. San Diego, CA, USA: IEEE. Recuperado a partir de https://doi.org/10.1109/CVPR.2005.177

Davison, A. J. (2003). Real-time simultaneous localisation and mapping with a single camera. *Proceedings of the IEEE International Conference on Computer Vision*, 2, 1403-1410. Recuperado a partir de https://doi.org/10.1109/iccv.2003.1238654

Deepika, N., & Sajith Variyar, V. V. (2017). Obstacle classification and detection for vision based navigation for autonomous driving. *2017 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2017*, 2017-Janua, 2092-2097. Recuperado a partir de https://doi.org/10.1109/ICACCI.2017.8126154

Durrant-Whyte, H., & Bailey, T. (2006). Simultaneous Localization and Mapping: Part I. *IEEE Robotics & Automation Magazine*, 13(2), 99-110. Recuperado a partir de https://doi.org/10.1182/blood.v80.3.688.bloodjournal803688

eLinux. (2018). Rpi Camera Module. Recuperado 1 de junio de 2020, a partir de https://elinux.org/Rpi_Camera_Module#Introduction

Engel, J., Stückler, J., & Cremers, D. (2015). Large-scale direct SLAM with stereo cameras. *IEEE International Conference on Intelligent Robots and Systems*, 2015-Decem, 1935-1942. Recuperado a partir de https://doi.org/10.1109/IROS.2015.7353631

Frese, U., Wagner, R., & Röfer, T. (2010). A SLAM Overview from a User's Perspective. *KI - Künstliche Intelligenz*, 24(3), 191-198. Recuperado a partir de https://doi.org/10.1007/s13218-010-0040-4

Gao, X., & Zhang, T. (2017). Unsupervised learning to detect loops using deep neural networks for visual SLAM system. *Autonomous Robots*, 41(1), 1-18. Recuperado a partir de https://doi.org/10.1007/s10514-015-9516-2

GeeksforGeeks. (2018). Object Detection vs Object Recognition vs Image Segmentation. Recuperado 27 de julio de 2020, a partir de https://www.geeksforgeeks.org/object-detection-vs-object-recognition-vs-image-segmentation/

Goyal, S., & Benjamin, P. (2014). Object Recognition Using Deep Neural Networks: A Survey, 1-7. Recuperado a partir de http://arxiv.org/abs/1412.3684

Griffin, G., Holub, A., & Perona, P. (2006). The Caltech 256. Recuperado 3 de septiembre de 2020, a partir de http://www.vision.caltech.edu/Image_Datasets/Caltech256/

Guo, Y., Bennamoun, M., Sohel, F., Lu, M., & Wan, J. (2014). 3D object recognition in cluttered scenes with local surface features: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11), 2270-2287. Recuperado a partir de https://doi.org/10.1109/TPAMI.2014.2316828

Hamzah, R. A., & Ibrahim, H. (2016). Literature survey on stereo vision disparity map algorithms. *Journal of Sensors*, 2016. Recuperado a partir de

https://doi.org/10.1155/2016/8742920

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. En *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 770-778). IEEE. Recuperado a partir de https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html

Hu, H., Gu, J., Zhang, Z., Dai, J., & Wei, Y. (2018). Relation Networks for Object Detection. En *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 3588-3597). Recuperado a partir de https://doi.org/10.1109/CVPR.2018.00378

Hu, J., Shen, L., & Sun, G. (2018). Squeeze-and-Excitation Networks. En *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 7132-7141). IEEE. Recuperado a partir de http://openaccess.thecvf.com/content_cvpr_2018/html/Hu_Squeeze-and-Excitation_Networks_CVPR_2018_paper.html

Intel Corporation. (2020). Tecnología Intel RealSense. Recuperado 22 de julio de 2020, a partir de https://www.intel.la/content/www/xl/es/architecture-and-technology/realsense-overview.html

IntoRobotics. (2013). Stereo Vision Cameras for Robots - Tutorials and Resources. Recuperado 22 de julio de 2020, a partir de https://www.intorobotics.com/fundamental-guide-for-stereo-vision-cameras-in-robotics-tutorials-and-resources/

Jafri, R., Ali, S. A., Arabnia, H. R., & Fatima, S. (2014). Computer vision-based object recognition for the visually impaired in an indoors environment: a survey. *Visual Computer*, 30(11), 1197-1222. Recuperado a partir de https://doi.org/10.1007/s00371-013-0886-1

Jones, D. (2013). Picamera. Recuperado 30 de junio de 2020, a partir de https://picamera.readthedocs.io/en/release-1.13/

Karras, T., Laine, S., & Aila, T. (2019). A style-based generator architecture for generative adversarial networks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June, 4396-4405. Recuperado a partir de https://doi.org/10.1109/CVPR.2019.00453

Kasyanov, A., & Engelmann, F. (2017). Keyframe-based visual-inertial online SLAM with

relocalization 2017 cited_5 KEYFRAME MAKALESİ.pdf. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (2).

Kim, Y. M., Theobalt, C., Diebel, J., Kosecka, J., Miscusik, B., & Thrun, S. (2009). Multi-view image and ToF sensor fusion for dense 3D reconstruction. *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops 2009*, 1542-1546. Recuperado a partir de https://doi.org/10.1109/ICCVW.2009.5457430

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep ConvolutionalNeural Networks. En *Advances in Neural Information Processing Systems 25 (NIPS 2012)* (pp. 1-9). Recuperado a partir de http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networ

Lategahn, H., Geiger, A., & Kitt, B. (2011). Visual SLAM for autonomous ground vehicles. *Proceedings - IEEE International Conference on Robotics and Automation*, 1732-1737. Recuperado a partir de https://doi.org/10.1109/ICRA.2011.5979711

Lee, J., Cho, Y., Nam, C., Park, J., & Kim, C. (2019). Efficient obstacle rearrangement for object manipulation tasks in cluttered environments. En *Proceedings - IEEE International Conference on Robotics and Automation* (Vol. 2019-May, pp. 183-189). IEEE. Recuperado a partir de https://doi.org/10.1109/ICRA.2019.8793616

Lemaire, T., Berger, C., Jung, I. K., & Lacroix, S. (2007). Vision-based SLAM: Stereo and monocular approaches. *International Journal of Computer Vision*, 74(3), 343-364. Recuperado a partir de https://doi.org/10.1007/s11263-007-0042-3

Levkovits-Scherer, D. S., Cruz-Vega, I., & Martinez-Carranza, J. (2019). Real-Time Monocular Vision-Based UAV Obstacle Detection and Collision Avoidance in GPS-Denied Outdoor Environments Using CNN MobileNet-SSD. En L. Martínez-Villaseñor, I. Batyrshin, & A. Marín-Hernández (Eds.), *Advances in Soft Computing* (pp. 613-621). Cham: Springer International Publishing. Recuperado a partir de https://link.springer.com/chapter/10.1007/978-3-030-33749-0_49

Li, W., & Xiong, R. (2019). Dynamical Obstacle Avoidance of Task- Constrained Mobile Manipulation Using Model Predictive Control. *IEEE Access*, 7, 88301-88311. Recuperado a partir de https://doi.org/10.1109/ACCESS.2019.2925428

Lin, T.-Y., Dollar, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017). Feature Pyramid Networks for Object Detection. En *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 2117-2125). IEEE. Recuperado a partir de https://openaccess.thecvf.com/content_cvpr_2017/html/Lin_Feature_Pyramid_Networks

_CVPR_2017_paper.html

Liu, C., Zheng, B., Wang, C., Zhao, Y., Fu, S., & Li, H. (2017). CNN-based vision model for obstacle avoidance of mobile robot. *MATEC Web of Conferences*, 139, 4-7. Recuperado a partir de https://doi.org/10.1051/matecconf/201713900007

Marín, R. (2020). ¿Qué es OpenCV? Instalación en Python y ejemplos básicos. Recuperado 28 de julio de 2020, a partir de https://revistadigital.inesem.es/informatica-y-tics/opencv/

Marr, D., & Poggio, T. (2013). A computational theory of human stereo vision. *Readings in Cognitive Science: A Perspective from Psychology and Artificial Intelligence*, 328, 534-547. Recuperado a partir de https://doi.org/10.1016/B978-1-4832-1446-7.50046-7

MathWorks, I. (2018). ¿Qué es el reconocimiento de objetos? Recuperado 22 de julio de 2020, a partir de https://la.mathworks.com/solutions/image-video-processing/object-recognition.html

Michelson, R. C. (2000). Autonomous navigation. *AccessScience*. Recuperado a partir de https://doi.org/10.1036/1097-8542.YB000130

Middlebury College. (2015). 2014 Stereo datasets with ground truth. Recuperado 21 de julio de 2020, a partir de https://vision.middlebury.edu/stereo/data/scenes2014/

Milz, S., Arbeiter, G., Witt, C., Abdallah, B., & Yogamani, S. (2018). Visual SLAM for automated driving: Exploring the applications of deep learning. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2018-June, 360-370. Recuperado a partir de https://doi.org/10.1109/CVPRW.2018.00062

Mobile Industrial Robots A/S. (2020). Frontpage. Recuperado 28 de julio de 2020, a partir de https://www.mobile-industrial-robots.com/en/

Mouragnon, E., Lhuillier, M., Dhome, M., Dekeyser, F., & Sayd, P. (2006). Real time localization and 3D reconstruction. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1, 363-370. Recuperado a partir de https://doi.org/10.1109/CVPR.2006.236

Mur-Artal, R., & Tardós, J. D. (2017). ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras. *IEEE Transactions on Robotics*, 33(5), 1255-1262. Recuperado a partir de https://doi.org/10.1109/TRO.2017.2705103

Newman, P., Cole, D., & Ho, K. (2006). Outdoor SLAM using visual appearance and laser ranging. *Proceedings - IEEE International Conference on Robotics and Automation*,

2006(May), 1180-1187. Recuperado a partir de https://doi.org/10.1109/ROBOT.2006.1641869

Nistér, D. (2005). Preemptive RANSAC for live structure and motion estimation. *Machine Vision and Applications*, 16(5), 321-329. Recuperado a partir de https://doi.org/10.1007/s00138-005-0006-y

Nistér, D., Naroditsky, O., & Bergen, J. (2004). Visual odometry. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1. Recuperado a partir de https://doi.org/10.1109/cvpr.2004.1315094

Nistér, D., & Stewénius, H. (2006). Scalable recognition with a vocabulary tree. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2, 2161-2168. Recuperado a partir de https://doi.org/10.1109/CVPR.2006.264

OpenCV team. (2020). About OpenCV. Recuperado 30 de junio de 2020, a partir de https://opencv.org/about/

Parkhi, O. M., Vedaldi, A., Zisserman, A., & Jawahar, C. V. (2012). Cats and dogs. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 3498-3505. Recuperado a partir de https://doi.org/10.1109/CVPR.2012.6248092

Prabhakar, G., Kailath, B., Natarajan, S., & Kumar, R. (2017). Obstacle detection and classification using deep learning for tracking in high-speed autonomous driving. *TENSYMP 2017 - IEEE International Symposium on Technologies for Smart Cities*, 3-8. Recuperado a partir de https://doi.org/10.1109/TENCONSpring.2017.8069972

Ramisa, A., Aldavert, D., Vasudevan, S., Toledo, R., & Lopez de Mantaras, R. (2015). Evaluation of Three Vision Based Object Perception Methods for a Mobile Robot. En *Household Service Robotics* (Vol. 68, pp. 303-337). Zhejiang University Press Co., Ltd. Recuperado a partir de https://doi.org/10.1016/B978-0-12-800881-2.00015-3

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). (YOLO) You Only Look Once. En *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 779-788). IEEE. Recuperado a partir de https://doi.org/10.1109/CVPR.2016.91

Rosebrock, A. (2018). OpenCV Face Recognition. Recuperado 28 de julio de 2020, a partir de https://www.pyimagesearch.com/2018/09/24/opencv-face-recognition/

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., … Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3), 211-252. Recuperado a partir de https://doi.org/10.1007/s11263-015-0816-y

Scharstein, D., Szeliski, R., & Zabih, R. (2001). A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Proceedings - IEEE Workshop on Stereo and Multi-Baseline Vision, SMBV 2001*, 47(1), 131-140. Recuperado a partir de https://doi.org/10.1109/SMBV.2001.988771

Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 1-14.

Singh, A. (2019). A Detailed Guide to the Powerful SIFT Technique for Image Matching (with Python code). Recuperado 23 de julio de 2020, a partir de https://www.analyticsvidhya.com/blog/2019/10/detailed-guide-powerful-sift-technique-image-matching-python/

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., … Rabinovich, A. (2015). Going Deeper with Convolutions. En *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 1-9). IEEE. Recuperado a partir de https://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Szegedy_Going_Deeper_With_2015_CVPR_paper.html

Tesla. (2020). Electric Cars, Solar & Clean Energy. Recuperado 28 de julio de 2020, a partir de https://www.tesla.com/

Thrun, S., & Montemerlo, M. (2006). The graph SLAM algorithm with applications to large-scale mapping of urban structures. *International Journal of Robotics Research*, 25(5-6), 403-429. Recuperado a partir de https://doi.org/10.1177/0278364906065387

Uijlings, J. R. R., Van De Sande, K. E. A., Gevers, T., & Smeulders, A. W. M. (2013). Selective search for object recognition. *International Journal of Computer Vision*, 104(2), 154-171. Recuperado a partir de https://doi.org/10.1007/s11263-013-0620-5

UW CSE vision faculty. (2017). Stereo and 3D Vision. Recuperado 20 de julio de 2020, a partir de https://courses.cs.washington.edu/courses/cse455/09wi/Lects/lect16.pdf

Viola, P., & Jones, M. (2001). Rapid Object Detection using a Boosted Cascade of Simple

Features. En *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001* (pp. 511-518). Kauai, HI, USA: IEEE. Recuperado a partir de https://doi.org/10.1109/CVPR.2001.990517

virt2real. (2019a). OpenCV and Depth Map on StereoPi tutorial. Recuperado 25 de junio de 2020, a partir de https://stereopi.com/blog/opencv-and-depth-map-stereopi-tutorial

virt2real. (2019b). StereoPi Crowdsupply Page. Recuperado a partir de https://www.crowdsupply.com/virt2real/stereopi

virt2real. (2020). StereoPi Wiki. Recuperado 10 de mayo de 2020, a partir de https://wiki.stereopi.com/index.php?title=Main_Page

Wang, Y., Chao, W. L., Garg, Di., Hariharan, B., Campbell, M., & Weinberger, K. Q. (2019). Pseudo-lidar from visual depth estimation: Bridging the gap in 3D object detection for autonomous driving. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June, 8437-8445. Recuperado a partir de https://doi.org/10.1109/CVPR.2019.00864

Yu, H., Hong, R., Huang, X., & Wang, Z. (2013). Obstacle detection with deep convolutional neural network. En *Proceedings - 6th International Symposium on Computational Intelligence and Design, ISCID 2013* (Vol. 1, pp. 265-268). Recuperado a partir de https://doi.org/10.1109/ISCID.2013.73

Zhang, G., Lee, J. H., Lim, J., & Suh, I. H. (2015). Building a 3-D line-based map using stereo SLAM. *IEEE Transactions on Robotics*, 31(6), 1364-1377. Recuperado a partir de https://doi.org/10.1109/TRO.2015.2489498

Zhang, J., Tai, L., Boedecker, J., Burgard, W., & Liu, M. (2017). Neural SLAM: Learning to Explore with External Memory. Recuperado a partir de http://arxiv.org/abs/1706.09520

Zhao, X., Cao, Z., Jia, Q., Pang, L., Yu, Y., & Tan, M. (2018). A vision-based robotic grasping approach under the disturbance of obstacles. En *Proceedings of 2018 IEEE International Conference on Mechatronics and Automation, ICMA 2018* (pp. 2175-2179). IEEE. Recuperado a partir de https://doi.org/10.1109/ICMA.2018.8484277

Zhi, S., Liu, Y., Li, X., & Guo, Y. (2018). Toward real-time 3D object recognition: A lightweight volumetric CNN framework using multitask learning. *Computers and Graphics (Pergamon)*, 71, 199-207. Recuperado a partir de https://doi.org/10.1016/j.cag.2017.10.007

Anexos

Anexo I. Tabla completa de distancias estimadas

Tabla detallada de la distancia de ruta crítica estimada para cada escena con sus distintos casos, y el error calculado.

Escenario de prueba		*Distancia estimada de [cm]*	*Distancia real dr [cm]*	*Error e = de - dr [cm]*	*Error porcentual absoluto [%]*
A	***a***	166	máxima	-	
	b	66	máxima	-	
	c	68	máxima		
	d	59	máxima	-	
C	***a***	43	50	-7	14.0%
	b	42	50	-8	16.0%
	c	42	50	-8	16.0%
	d	43	50	-7	14.0%
D	***a***	64	45	19	42.2%
	b	55	45	10	22.2%
	c	54	45	9	20.0%
E	***a***	65	70	-5	7.1%
	b	64	70	-6	8.6%
	c	68	70	-2	2.9%
F	***a***	51	55	-4	7.3%
	b	41	55	-14	25.5%
	c	40	55	-15	27.3%
	d	48	55	-7	12.7%
G	***a***	166	118	48	40.7%
	b	65	118	-53	44.9%
	c	86	118	-32	27.1%
	d	88	118	-30	25.4%
H	***a***	166	75	91	121.3%
	b	99	75	24	32.0%
	c	81	75	6	8.0%
I	***a***	91	70	21	30.0%
	b	66	70	-4	5.7%
	c	64	70	-6	8.6%
J	***a***	69	100	-31	31.0%
	b	66	100	-34	34.0%
	c	76	100	-24	24.0%
				Error promedio	24.8%
				Exactitud	**75.2%**

Anexo II. Tabla completa de los resultados de la detección de obstáculos

Tabla detallada de los resultados obtenidos en la detección de obstáculos para cada escena y sus casos, junto con los estados de referencia. La fila coloreada corresponde a la captura donde no se detectó correctamente el obstáculo.

Escenario de prueba		*Obstáculo detectado [Si / No]*	*Obstáculo existe [Si / No]*
A	***a***	No	No
	b	Si	Si
	c	Si	Si
	d	Si	Si
C	***a***	Si	Si
	b	No	Si
	c	Si	Si
	d	Si	Si
D	***a***	No	No
	b	Si	Si
	c	Si	Si
E	***a***	No	No
	b	Si	Si
	c	Si	Si
F	***a***	No	No
	b	Si	Si
	c	Si	Si
	d	Si	Si
G	***a***	No	No
	b	Si	Si
	c	Si	Si
	d	Si	Si
H	***a***	No	No
	b	Si	Si
	c	Si	Si
I	***a***	No	No
	b	Si	Si
	c	Si	Si
J	***a***	No	No
	b	Si	Si
	c	Si	Si
Cuenta (Si)		**22**	**23**
		Exactitud	***95.7%***

Anexo III. Tabla completa de los resultados de la clasificación de obstáculos

Tabla detallada de los resultados obtenidos en la clasificación de obstáculos para cada escena y sus casos, junto con las categorías reales y la cuenta de obstáculos clasificados en cada categoría.

Escenario de prueba		*Categoría predicha [0:Común / 1:Crítico]*	*Categoría real [0:Común / 1:Crítico]*
A	***b***	1	0
	c	1	1
	d	1	1
C	***a***	1	0
	c	1	1
	d	1	1
D	***b***	0	0
	c	1	1
E	***b***	1	0
	c	1	1
F	***b***	0	0
	c	1	1
	d	1	1
G	***b***	0	0
	c	1	1
	d	1	1
H	***b***	1	0
	c	1	1
I	***b***	1	0
	c	1	1
J	***b***	0	0
	c	1	1
Cuenta (0)		**4**	**9**
Cuenta (1)		**18**	**13**

Anexo IV. Artículo de investigación

Detección y clasificación de obstáculos mediante visión estéreo en la toma inteligente de decisiones para robots autónomos en espacios reducidos

José Daniel Báez Maldonado

Universidad Internacional de la Rioja, Logroño (España)

22 de septiembre, 2020

Resumen

Mientas aumenta el uso de robots de navegación autónoma dentro de ambientes compartidos con humanos, se vuelve más evidente la necesidad de mejorar las medidas de seguridad que éstos integran para que su movilidad no represente un riesgo para personas y animales. Este trabajo presenta una herramienta de software para la detección y clasificación de obstáculos cercanos mediante algoritmos de visión artificial, orientado a la toma de decisiones que protejan la integridad física de niños y mascotas que se encuentren en el entorno de robots autónomos. Se utilizan mapas de profundidad generados por una cámara estéreo para identificar la ruta crítica y los obstáculos presentes, junto con una red neuronal convolucional para su clasificación. Una etapa final de toma de decisiones recopila la información obtenida para sugerir las acciones más adecuadas de acuerdo a cada caso, explorando la posibilidad de manipular los objetos no-críticos en caso de ser necesario.

Palabras Clave

Aprendizaje profundo, detección, navegación autónoma, obstáculo, visión estéreo.

I. Introducción

El uso de robots "inteligentes" con navegación autónoma se ha expandido notablemente durante los últimos años, hasta convertirse en herramientas cotidianas de trabajo en distintas aplicaciones que incluyen desde aspiradoras de uso doméstico hasta plataformas transportadoras de material para uso industrial. Mientras aumenta la popularidad de este tipo de robots también se vuelve más evidente la necesidad de mejorar las medidas de seguridad que integran, con el fin de salvaguardar la integridad física de las personas y animales que forman parte de los entornos donde éstos se desplazan.

Una forma de dar solución a esta problemática es la detección de dichos individuos potencialmente vulnerables, y así evitar la ejecución de acciones que puedan suponer un riesgo para éstos. Este reconocimiento es posible con el uso de clasificadores entrenados mediante técnicas de aprendizaje automático y visión computarizada, y puede guiar a tomar las decisiones adecuadas en base a un razonamiento lógico sencillo; contemplando la posibilidad de manipular los obstáculos comunes pequeños.

Este trabajo está compuesto por tres etapas principales: identificación de ruta crítica y detección de obstáculos, clasificación, y toma de decisiones. Se proporciona un método de detección de los objetos cercanos al robot, identificando a los niños y mascotas mediante un modelo inteligente; con el fin de realizar las acciones adecuadas que no representen un peligro a su seguridad. Para lograr esto, se utilizan mapas de profundidades generados por visión estéreo, en conjunto con un modelo de clasificación de imágenes entrenado mediante técnicas de aprendizaje profundo.

Adicionalmente, se explora la posibilidad de manipulación de objetos no-críticos en caso de ser necesario. La implementación y pruebas se realizaron en una tarjeta *StereoPi*, que permite la visión estereoscópica gracias a su compatibilidad de conexión de dos cámaras con el módulo de cómputo *Raspberry Pi*. Estas pruebas se llevaron a cabo en distintos ambientes de una casa familiar con objetos comunes del hogar.

Este artículo está organizado de la siguiente manera:

El capítulo II realiza un repaso de los trabajos destacados en el estado del arte referente a la navegación autónoma, evasión de obstáculos, y la aplicación de visión estéreo para localización y mapeo, junto con el uso de técnicas de inteligencia artificial y cámaras estereoscópicas para el reconocimiento y manipulación de objetos.

Los objetivos y metodología que direccionaron este trabajo se detallan en el capítulo III, mientras que el capítulo IV detalla las etapas y algoritmos utilizados de la contribución presentada.

La evaluación y resultados obtenidos se muestran en el capítulo V, junto con la discusión de los mismos en el capítulo VI. Finalmente, el capítulo VII recopila las conclusiones del trabajo realizado.

II. Estado del Arte

En las últimas décadas, la robótica ha presentado un gran crecimiento; poco a poco, los robots se están convirtiendo en compañeros cotidianos de las personas. La interacción y convivencia humano-máquina se vuelve evidente conforme el continuo desarrollo permite la creación de robots más "inteligentes". Un gran ejemplo son los robots móviles con navegación autónoma. Una gran variedad de sistemas y características se combinan con el fin de proporcionar dicho grado de inteligencia y autonomía, que les permite tomar decisiones y ejecutar acciones de acuerdo a sus funcionalidades y su entorno. Estos sistemas incluyen desde operaciones sencillas

de sensado hasta algoritmos complejos de procesamiento de datos e imágenes.

Visión estéreo

El término "**visión estéreo**" se define como el proceso que combina múltiples imágenes de una escena para extraer información geométrica tridimensional. Su versión más común y sencilla utiliza solamente dos imágenes, se conoce también como visión binocular y está directamente inspirada en el sistema visual de los humanos y animales [1]. Gracias a la ubicación de los ojos en el rostro humano, la información captada y enviada al sistema nervioso central son dos imágenes similares del entorno, tomadas desde dos puntos cercanos en el mismo nivel horizontal. La posición de un objeto en una imagen difiere respecto a la otra, dependiendo de la distancia a la que se encuentra del observador. El cerebro es capaz de medir esta disparidad y usarla para estimar la profundidad a la que se encuentra cada objeto observado [2].

Los algoritmos de visión estéreo, también conocida como visión estereoscópica, buscan modelar computacionalmente esta función que es ejecutada de manera natural por las personas. Para lograrlo, Marr & Poggio [2] establecen tres pasos principales: se selecciona un punto o región pequeña en una de las imágenes; el mismo punto debe ser identificado en la otra imagen; y finalmente se mide la disparidad entre los puntos correspondientes. Estos pasos se deben realizar para todos los puntos que se consideren necesarios para una buena interpretación del entorno. Dependiendo de los requerimientos de la aplicación, se puede calcular la profundidad a partir de la disparidad encontrada.

La **disparidad** se define como la diferencia de localización en imagen de un mismo punto tridimensional, cuando es proyectado bajo la perspectiva de dos cámaras distintas [3]. Si la localización horizontal de este punto en la vista izquierda es x_{izq} y en la vista derecha es x_{der}, la disparidad d se obtiene mediante la diferencia de ambos valores, como se indica a continuación:

$$d = x_{izq} - x_{der}$$

Sin embargo, conocer la disparidad de un solo punto no es suficiente para tener una buena comprensión de la tridimensionalidad de la escena. Es necesario generar un mapa de disparidad que pueda representar la totalidad, o una porción significativa, del entorno observado. Un esquema de los pasos involucrados en el desarrollo de un algoritmo para visión estéreo es presentado por Scharstein, Szeliski, & Zabih [4], como se indica en la Figura 1.

Fig. 1. Pasos en el desarrollo de algoritmos de visión estéreo.

Reconocimiento de objetos

El **reconocimiento** de objetos, o **clasificación** de imágenes, es una técnica de visión computarizada que identifica si un objeto específico está presente o no en una imagen o video [5]. Esto es posible gracias al entrenamiento de algoritmos de aprendizaje automático, a partir de grandes bases de datos de imágenes y sus etiquetas que indican la presencia del objeto de interés. Su objetivo es obtener información que permita a las máquinas entender el contenido de una imagen. La utilidad de esta información ha logrado que el reconocimiento de objetos se convierta en un recurso clave en vehículos autónomos, la inspección industrial, visión robótica, entre otras aplicaciones variadas.

Entre los algoritmos de reconocimiento de objetos que utilizan técnicas de aprendizaje automático clásico, Ramisa et al. [6] mencionan en su libro tres métodos destacados: el algoritmo SIFT, la bolsa de características[7], y clasificadores simples en cascada [8]. Por su parte, el aprendizaje profundo ha revolucionado la manera de dar solución a muchos de los problemas de visión artificial, siendo el reconocimiento y la detección de objetos de los más destacados [9]. Desde el éxito obtenido por AlexNet [10] al utilizar una red neuronal convolucional (CNN) para clasificación de imágenes en el reto ImageNet [11], las redes convolucionales se convirtieron en el método más utilizado y explorado para el procesamiento de imágenes y videos en el campo de la inteligencia artificial. Una de sus principales ventajas es que la extracción de características es realizada automáticamente dentro de la red, facilitando su implementación y generalización [12].

La **detección** de objetos combina el reconocimiento de objetos con la identificación de su localización en la imagen. Muchos algoritmos de reconocimiento de objetos han sido adaptados para convertirse en detectores capaces de diferenciar varios objetos de diferentes clases y la localización de cada uno de ellos. Entre los sistemas de detección más conocidos se encuentra *YOLO* [13], que utiliza una única red neuronal para obtener la localización de los objetos y la clase a la que pertenecen.

Además de la detección de imágenes 2D, existen algoritmos de detección que utilizan información tridimensional [14] como las redes neuronales convolucionales 2.5D que utilizan las profundidades obtenidas por cámaras RGB-D o estéreo como un canal adicional; o las redes convolucionales 3D [15].

Navegación autónoma

La navegación autónoma permite que un vehículo o robot móvil sea capaz de planificar su ruta y ejecutarla sin la intervención de un operador humano [16]. Para lograrlo se hace uso de la información de su entorno capturada a través de una gran variedad de sensores, y de los datos almacenados, ya sea localmente o en servidores relacionados. En conjunto con campos relacionados como la inteligencia artificial y la visión computarizada, los algoritmos de navegación autónoma han ganado mucho reconocimiento al lograr poner en movimiento desde pequeños robots y drones, hasta automóviles comerciales [17] y robots de uso industrial [18].

Una parte fundamental en la navegación autónoma es el problema de **localización y mapeo simultáneo**, más conocido en la literatura como *SLAM*, el cual ha sido abordado desde distintos métodos desde su origen en 1986 [19]. Entre los sistemas de localización que utilizan cámaras, conocidos como *visual-SLAM*, se presentan dos grupos principales: los que utilizan una única cámara para realizar el mapeo del entorno, llamados *SLAM monocular* [20]–[23]; y los que utilizan más de una cámara (generalmente dos), conocidos como *SLAM estéreo* [24]–[26]. Las redes neuronales también han sido aplicadas a la navegación y localización mediante propuestas como *Neural SLAM* [27], entre otras [28]–[30].

Otra de las funciones que debe incluir una máquina para desplazarse de manera autónoma es la capacidad de detectar los obstáculos que se presenten, y evadirlos para no colisionar contra ellos. "La habilidad de los robots móviles para navegar y evadir obstáculos es un indicador importante de la inteligencia del robot" [31]. Muchas de las propuestas utilizan redes neuronales para la detección de obstáculos [31]–[34], destinados para vehículos terrestres y aéreos o *UAV*s [35].

III. Objetivos y Metodología

Objetivo general

Desarrollar un software de detección y clasificación de obstáculos cercanos mediante algoritmos de visión artificial, orientado a la toma de decisiones que protejan la integridad física de niños y mascotas que se encuentren en el entorno de robots autónomos, utilizando técnicas de aprendizaje profundo y mapas de profundidad generados por una cámara estéreo.

Objetivos específicos

- Realizar un análisis del estado del arte en sistemas de reconocimiento de obstáculos para robots de navegación autónoma y visión estéreo.
- Localizar los obstáculos que se encuentren más cercanos a las cámaras e identificar si existe el espacio suficiente para evadirlos, a partir de un mapa de profundidades.
- Clasificar los obstáculos detectados en común o crítico, mediante el entrenamiento de una red neuronal convolucional profunda.
- Tomar decisiones inteligentes que determinen las acciones que deben ejecutarse de manera adecuada y segura, en base a toda la información adquirida.
- Implementar el sistema en una tarjeta StereoPi y evaluar los resultados obtenidos ubicándola en varios escenarios de prueba.

Metodología

Con el fin de alcanzar los objetivos del presente desarrollo, se utilizó una de las metodologías más utilizadas para proyectos de inteligencia artificial, llamada CRISP-DM por sus siglas en inglés para "CRoss-Industry Standard Process for Data Mining". El proceso que se indica en la CRISP-DM consta de 6 fases:

1. Entendimiento del tema: establece los objetivos del proyecto en base a lo que se busca alcanzar e identifica los requerimientos del desarrollo.
2. Entendimiento de la información: recolección y captura de datos, y análisis detallado de los mismos.
3. Preparación de datos: operaciones necesarias para que la información inicial sea útil y entendible por los algoritmos utilizados posteriormente.
4. Modelamiento: describe las operaciones realizadas durante el procesamiento específico de la información ya tratada, y el entrenamiento del modelo de clasificación.
5. Evaluación: pruebas y métricas necesarias para determinar en qué grado los resultados obtenidos por la etapa de procesamiento son satisfactorios o no.
6. Producción: guardado y serialización de la red neuronal entrenada.

IV. Contribución

La contribución presentada consiste en un software de identificación y clasificación de obstáculos, orientado a la toma de decisiones para robots de navegación autónoma en entornos de espacio reducido. El desarrollo ha sido pensado para su implementación en tarjetas Raspberry Pi, debido a que son altamente usadas en aplicaciones de investigación en robótica. Se utilizan dos cámaras a modo de cámara estereoscópica para el reconocimiento del tipo de obstáculo y la estimación de profundidades. Los datos de profundidad permiten al robot identificar los obstáculos cercanos, y saber si existe el espacio necesario para evadir el obstáculo; caso contrario, clasificarlo para conocer si es posible manipularlo para continuar su navegación. Esta herramienta desarrollada en Python hace uso de algunas funciones proporcionadas por la librería OpenCV y una red neuronal convolucional profunda para clasificar los obstáculos en dos tipos: comunes y críticos, siendo los críticos personas, principalmente niños pequeños debido al rango de visibilidad del hardware, o mascotas. De esta forma, el robot podrá reconocer si el obstáculo es manipulable (común) o no manipulable (crítico), para "decidir" su siguiente paso.

Captura y análisis de datos

Las capturas estéreo consisten en imágenes RGB con resolución de 1280x480 píxeles, obtenidas por una cámara estereoscópica casera, construida con los elementos incluidos en el paquete inicial (Starter Kit) de StereoPi, que está compuesta por dos cámaras para Raspberry Pi de 5 megapíxeles, ubicadas con una separación horizontal de 6.5 cm y orientadas en la misma dirección. Estas imágenes son fotografías del entorno, donde se observan principalmente paredes, obstáculos y objetos varios alejados del robot a distancias entre 0.5 y 4 metros.

La lectura de las imágenes se realiza mediante el uso de la librería "picamera" [36], especializada para el uso de módulos de cámara junto con tarjetas Raspberry Pi. Se obtiene una captura continua de imágenes estéreo a través del puerto de video. Cada cuadro obtenido es una imagen a color que contiene la vista de la cámara izquierda y la vista de la cámara derecha ubicadas una al lado de la otra. La resolución 1280x480 significa que la imagen estéreo está compuesta por dos imágenes de 640x480 píxeles, siendo esta última la resolución individual de las cámaras. Se realizó la calibración estéreo de las cámaras, utilizando un patrón de ajedrez 7x9.

Entrenamiento del modelo de clasificación

Se realizó el entrenamiento de un modelo de aprendizaje profundo para el reconocimiento de los obstáculos a partir de imágenes. Este modelo consiste de una red neuronal convolucional de clasificación binaria de los obstáculos en dos categorías: comunes, o críticos. Los obstáculos comunes son artículos variados que podrían encontrarse en el camino y podrían ser manipulados sin representar riesgos, como juguetes, calzado, útiles escolares o de oficina, herramientas, entre otros. Los obstáculos críticos son personas como bebés o niños pequeños, y animales domésticos como perros o gatos. Una interacción demasiado cercana entre éstos y el robot podría ser riesgosa, por lo que se tomarán acciones que no representen peligro alguno.

Selección y análisis de bases de datos

Las imágenes utilizadas para el entrenamiento han sido obtenidas de distintas bases de datos junto con imágenes propias y se han seleccionado las más relevantes de acuerdo a los obstáculos que se pueden encontrar en entornos cerrados. Éstas corresponden a fotografías de objetos variados que han sido agrupadas en las dos categorías de interés para este trabajo: comunes o críticos.

Las imágenes de mascotas se obtuvieron de la base de datos The Oxford-IIIT Pet Dataset [37]. Se utilizaron imágenes de rostros humanos obtenidas de la base de datos Flickr-Faces-HQ Dataset [38], además de imágenes de bebés obtenidas en Google y fotografías propias de los objetos utilizados como obstáculos críticos en las pruebas, desde distintas perspectivas.

Las imágenes de obstáculos comunes fueron seleccionadas de la base de datos Caltech 256 [39] y se añadieron fotografías

propias de algunos objetos utilizados como obstáculos críticos y de distintos ambientes del hogar utilizados como escenarios de prueba.

En total se compilaron 3140 imágenes de la categoría "obstáculos críticos", y 2055 imágenes de la categoría "obstáculos comunes", según el detalle en la Tabla I. De éstas, 4000 fueron utilizadas para el entrenamiento del modelo de clasificación y 1195 para su evaluación.

TABLA I

IMÁGENES UTILIZADAS PARA EL ENTRENAMIENTO DEL MODELO CNN

Tipo de obstáculo	*Número de imágenes*	*Base de datos de origen*	*Contenido de las imágenes*	*Resolución [píxeles]*
1: Crítico	2000	*The Oxford-IIIT Pet Dataset*	Perros y gatos	Variada
	1000	*Flickr-Faces-HQ Dataset*	Rostros de personas	128x128
	42	*Google*	Bebés	Variada
	98	*Fotografías propias*	Objetos críticos de prueba	4000x3000 3000x3000
0: Común	2000	*Caltech 256*	Objetos variados	Variada
	55	*Fotografías propias*	Objetos comunes de prueba	3000x3000
Total	**5195**			

Preprocesamiento de datos

Las imágenes recopiladas deben ser preparadas y normalizadas para un entrenamiento adecuado de la red neuronal. Ya que las imágenes originales tienen dimensiones variadas, han sido redimensionadas a un tamaño estándar de 64 x 64 píxeles.

Como siguiente paso, se realiza una ecualización de histograma para realzar el contraste de las imágenes. Debido a que no es posible llevar a cabo esta operación directamente sobre imágenes RGB, éstas son primero transformadas al modelo HSV para realizar la ecualización de histograma sobre el canal correspondiente al brillo. Posteriormente, la imagen resultante es transformada nuevamente al modelo RGB. Esta operación mejora la visualización y entendimiento de las imágenes y capturas oscuras que inicialmente dificultan la identificación de los objetos que se encuentran en ellas. Finalmente, las imágenes son normalizadas a valores 0 y 1.

El tipo de objeto correspondiente a cada imagen fue asignado a una variable categórica de longitud 2, donde el índice 0 corresponde a la clase "obstáculo común", y el índice 1 corresponde a la clase "obstáculo crítico".

Entrenamiento

El entrenamiento del modelo de aprendizaje profundo fue realizado en la interfaz *Keras* para python utilizando *TensorFlow* como backend. Se generó una red neuronal convolucional de aproximadamente 10 millones de parámetros con la arquitectura detallada en la Figura 2.

El modelo está compuesto por 4 capas convolucionales de 32, 64, 128, y 256 filtros, correspondientemente. El tamaño de filtros de la primera de estas capas es 5x5, mientras que todas las demás poseen filtros de dimensiones 3x3. Las 2 capas finales son capas densas, una de 64 neuronas y la capa de salida con 2 neuronas, de acuerdo al número de clases. La función de activación para todas las capas ocultas es la unidad lineal rectificada ReLU. Debido a que la salida corresponde a una variable categórica, la capa final utiliza la función de activación Softmax.

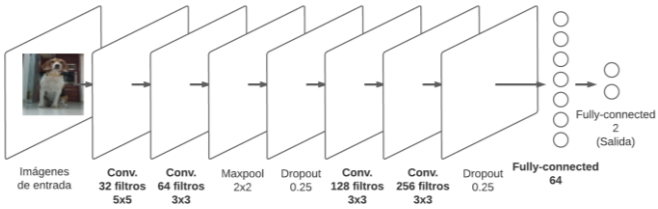


Fig. 2. Arquitectura de la red neuronal convolucional entrenada.

Para la regularización de la red se utilizó una capa MaxPooling después de la segunda capa convolucional y dos capas Dropout ubicadas después de la capa MaxPooling y después de la última capa convolucional, respectivamente.

La función de pérdida más adecuada para problemas de clasificación con salida de tipo categórica es la función de entropía categórica cruzada (categorical cross-entropy). Se utilizó esta función de pérdida con el optimizador AdaDelta para el entrenamiento del modelo en 200 épocas con 64 imágenes por lote (batch). El 20% de los datos de entrenamiento fueron utilizados como conjunto de validación durante el entrenamiento.

Serialización del modelo entrenado

Finalmente, el grafo de la red neuronal ya entrenada fue congelado y guardado, para de esta manera poder exportar el modelo completo al formato *protobuf*, incluyendo su arquitectura y sus pesos. Este archivo permitirá la utilización a futuro del modelo de clasificación desde la StereoPi mediante las funciones incluidas en la librería OpenCV, sin la necesidad de realizar en ella la instalación de TensorFlow.

Detección de obstáculos

Las imágenes capturadas por la StereoPi son preparadas para poder ser procesadas por el algoritmo generador de mapas de profundidad, mediante las siguientes operaciones de preprocesamiento: escalado y separación, conversión a escala de grises, y rectificación.

Preprocesamiento

Cada imagen estéreo obtenida es escalada con un factor de 0.5 antes de continuar con la ejecución de las demás operaciones con el fin de disminuir el tiempo de procesamiento, resultando en una resolución de 640x240 píxeles. Con el mismo propósito, las capturas escaladas son transformadas a la escala monocromática de grises. Posteriormente, la vista izquierda y derecha son separadas y rectificadas usando la librería para Python *stereovision*. El resultado es un nuevo par de imágenes: la vista izquierda rectificada y la vista derecha rectificada, las cuales están listas para ser usadas en la generación del mapa de disparidad.

Mapa de disparidad y profundidad

El mapa de disparidades de la escena observada por las cámaras es generado mediante la clase StereoBM (del inglés "Stereo Block Matching") de la librería OpenCV, a partir las imágenes rectificadas anteriormente. Al realizar el cálculo de las disparidades, esta función ejecuta un algoritmo de coincidencia de bloques para hallar la correspondencia entre la vista izquierda y derecha de la imagen estéreo, utilizando los siguientes parámetros previamente establecidos:

- SADWindowSize: 5
- minDisparity: -20
- numberOfDisparities: 80
- preFilterCap: 29
- preFilterSize: 5

- speckleRange: 4
- speckleWindowSize: 100
- textureThreshold: 200
- uniquenessRatio: 3

Una operación de escalado es aplicada a la matriz de disparidades para obtener valores entre 0 y 255 que podrán ser visualizados como un mapa de profundidades, donde los objetos más cercanos se graficarán con color rojo y los más alejados con azul.

Los valores de profundidad son obtenidos a partir de las disparidades mediante la ecuación:

$$Z = 8350 \,/\, (d - 65) \quad [cm]$$

Donde:

- Z es la profundidad en centímetros
- d es el valor de disparidad

Sin embargo, para disminuir el número de operaciones matriciales, la mayor parte de las tareas de procesamiento utilizan el mapa de disparidades. Los valores de profundidad se estiman únicamente para puntos específicos donde esta transformación sea necesaria. Debido a que este trabajo está orientado a la navegación en entornos de espacio reducido, y para una mejor identificación de los objetos de interés, solamente se consideran los valores de disparidad correspondientes a profundidades entre 0,5 y 1,5 metros. Los valores que se encuentran fuera de este rango son sustituidos por cero e ignorados por los siguientes pasos.

El mapa de disparidades obtenido es finalmente afinado mediante una operación de apertura y una de clausura, eliminando ruidos pequeños y dando mayor solidez a los objetos.

Ruta crítica y obstáculos

Se analiza un área parcial del mapa de profundidades, ubicada en la zona inferior, que ocupa el 20 por ciento de la altura total, donde se visualizan todos los obstáculos que se puedan encontrar delante de las cámaras en el rango especificado previamente.

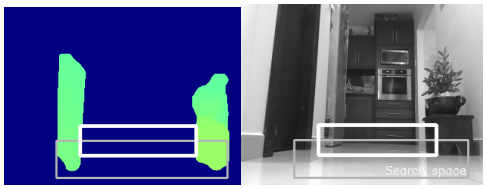


Fig. 3. Ruta crítica identificada en el mapa de profundidades.

Mediante una búsqueda exhaustiva con pasos de s=4 píxeles, se evalúa el espacio horizontal existente entre los segmentos con mayor disparidad de cada fila analizada. Este espacio representa la distancia entre los objetos más cercanos a la cámara. Se almacena temporalmente la mayor distancia local encontrada en la fila correspondiente, para posteriormente seleccionar la menor de las distancias locales como ruta crítica. De esta manera se logra identificar la ruta disponible más espaciosa, y conocer la sección más estrecha de la misma, como se observa en la Figura 3.

A continuación, los obstáculos son detectados mediante la extensión de la misma secuencia hacia los extremos. El tamaño de los obstáculos es filtrado para evitar falsos positivos.

La ruta crítica identificada, es transformada a unidades de distancia mediante la siguiente ecuación:

$$\Delta Y \;=\; dpix \;*\; Z \;* 0.0036 - 0.82$$

Donde:

- ΔY es la distancia entre objetos, medida en cm
- $dpix$ es la distancia entre objetos, medida en pixeles

Esta medida es comparada con el ancho del robot multiplicado por un factor de seguridad de 1,2. El resultado de esta comparación indicará si el espacio es suficiente para que el robot pueda movilizarse a través del mismo.

Clasificación de obstáculos

Cuando un obstáculo es detectado, se selecciona una porción rectangular de la captura correspondiente a la ubicación del obstáculo detectado para ser clasificada por el modelo entrenado. Este segmento es preprocesado de la misma manera que las imágenes de entrenamiento. Dicha imagen es categorizada por la red neuronal convolucional guardada en dos posibles tipos: común, o crítico. El espacio de búsqueda, la ruta crítica, y los obstáculos junto a su categoría son dibujados en cada captura como se muestra en la Figura 4.

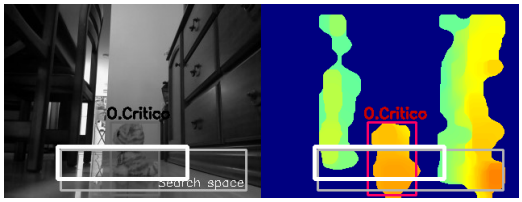


Fig. 4. Captura y mapa de profundidades etiquetados.

Módulo de decisiones

La combinación de los resultados obtenidos en cada una de las fases anteriores permite elegir la acción que se considere más adecuada. En los casos en que el obstáculo detectado en la vía no es crítico, el robot podrá manipularlo para continuar su navegación sin mayores cambios de ruta. Por otro lado, en los casos en que el obstáculo es una persona o mascota, el robot tomará acciones alternativas como detenerse y emitir una alarma sonora hasta que ésta le permita el paso, o caso contrario buscar una ruta distinta. Las acciones posibles son:

- Espacio suficiente: Continuar ruta
  (Obstáculo no detectado, espacio suficiente)
- Evadir obstáculo y continuar
  (Obstáculo común, espacio suficiente)
- Evadir y proceder con precaución
  (Obstáculo crítico, espacio suficiente)
- Manipular obstáculo para poder continuar
  (Obstáculo común, espacio parcial insuficiente)
- Emitir alerta sonora y esperar
  (Obstáculo crítico, espacio parcial insuficiente)
- Cambio de ruta
  (Espacio total insuficiente)

V. Evaluación y Resultados

Se realizaron pruebas en 9 escenarios distintos, correspondientes a zonas comunes del hogar, capturados bajo distintas circunstancias: sin obstáculo (Figura 5), con la presencia de un obstáculo común (Figura 6), y con la presencia de un obstáculo crítico (Figura 7). Los objetos utilizados como obstáculos son artículos domésticos o de oficina que podrían encontrarse casualmente sobre el suelo como pelotas, juguetes, calzado, esferográficos, etc. En representación de los obstáculos críticos correspondientes a mascotas y personas (principalmente niños), algunas capturas incluyen un perro y dos juguetes: un peluche de gato, y un muñeco de bebé.

Fig. 5. Escena E sin obstáculo.

Las imágenes fueron capturadas por la misma StereoPi que se utilizó durante el desarrollo de este trabajo, posicionada de forma que los lentes de las cámaras se encuentren a 7,5 cm de altura y su línea de visión central forme un ángulo de elevación de 15 grados con respecto al suelo. Esta configuración pretende simular la ubicación que las cámaras tendrían en un robot móvil, y permite optimizar el rango de visión para detectar desde objetos pequeños hasta objetos de gran tamaño, ya sea que se encuentren cerca o lejos de la StereoPi. El ancho de robot ingresado como ejemplo en el software fue de 20 cm.

Fig. 6. Escena F con obstáculo común.

De cada uno de los escenarios, enumerados con letras en orden alfabético de la B hasta la K, se obtiene la siguiente información:

- Distancia total de la ruta crítica [cm]
- Obstáculo detectado [Si/No]
- Distancia parcial por obstáculo [cm]
- Tipo de obstáculo (clasificación) [Común/Crítico]
- Decisión resultante (acción sugerida)

Todos estos resultados fueron recopilados para ser evaluados mediante distintas métricas.

Fig. 7. Escena C con obstáculo crítico.

Exactitud en la ruta crítica

Los resultados obtenidos por la etapa de identificación de la ruta crítica y estimación de distancias se detallan en la Tabla II. La distancia estimada promedio de cada escena corresponde a la media aritmética de las distancias obtenidas por los distintos casos de una misma escena.

TABLA II

ERROR EN LA ESTIMACIÓN DEL ESPACIO DE RUTA CRÍTICA

Escena de prueba	*Distancia estimada promedio: de [cm]*	*Distancia real: dr [cm]*	*Error: e = de - dr [cm]*	*Error porcentual absoluto [%]*
A	89.8	máxima	-	-
C	42.5	50	-7.5	15.0%
D	57.7	45	12.7	28.1%
E	65.7	70	-4.3	6.2%
F	45.0	55	-10.0	18.2%
G	101.3	118	-16.8	14.2%
H	115.3	75	40.3	53.8%
I	73.7	70	3.7	5.2%
J	70.3	100	-29.7	29.7%
			Error promedio	21.3%
			Exactitud	**78.7%**

Porcentaje de error producido por la etapa de estimación de ruta crítica en las escenas de prueba, y la exactitud resultante.

Rendimiento de la detección de obstáculos

La evaluación de los resultados obtenidos durante la etapa de detección de obstáculos está caracterizada por la matriz de confusión detallada en la Tabla III.

TABLA III

MATRIZ DE CONFUSIÓN DE LA ETAPA DE DETECCIÓN DE OBSTÁCULOS

Detectado:	*Real: Sin obstáculos*	*Real: Con obstáculos*	*Total*
Sin obstáculos	8	1	9
Con obstáculos	0	22	22
Total	8	23	31

La matriz de confusión especifica el número de objetos clasificados de manera correcta e incorrecta.

Se calcula la exactitud y precisión a partir de la matriz de confusión.

$$\boldsymbol{Exactitud} = \frac{22+8}{22+8+0+1} = 0.9677 = \mathbf{96.8\%}$$

$$\boldsymbol{Precision} = \frac{22}{22+0} = 1 = \mathbf{100\%}$$

Solamente uno de los 31 casos evaluados fue clasificado de manera incorrecta, logrando una exactitud mayor al 95%. Este ejemplo corresponde a un obstáculo que no fue percibido debido a su pequeño tamaño. No se produjo ningún falso positivo, alcanzando una precisión del 100%. Finalmente, se puede sintetizar la evaluación de la detección de obstáculos mediante una medida F1 = 0.98.

$$\boldsymbol{F1} = \frac{2 * precision * recall}{precision + recall} = \frac{2 * 1 * 0.957}{1 + 0.957} = \mathbf{0.98}$$

Rendimiento de la clasificación de obstáculos

En la Tabla IV se detallan los resultados obtenidos por la clasificación de obstáculos realizada por la red neuronal convolucional entrenada. Se han tomado en cuenta, para esta evaluación, solamente los 22 casos en los que se detectaron obstáculos.

TABLA IV

MATRIZ DE CONFUSIÓN DE LA ETAPA DE CLASIFICACIÓN DE OBSTÁCULOS

Categoría predicha:	*Categoría real: Común*	*Categoría real: Crítico*	*Total*
Común	4	0	4
Crítico	5	13	18
Total	9	13	22

La matriz de confusión especifica el número de casos detectados de manera correcta e incorrecta.

Se calcula la exactitud y sensibilidad a partir de la matriz de confusión.

$$\boldsymbol{Exactitud} = \frac{13+4}{13+4+5+0} = \frac{17}{23} = 0.7391 = \mathbf{73.9\%}$$

$$\boldsymbol{Sensibilidad} = \frac{22}{22+0} = 1 = \mathbf{100\%}$$

Se puede observar que 5 de los 9 obstáculos comunes fueron incorrectamente clasificados como críticos. Resultando en una exactitud de 74%. Sin embargo; todos los obstáculos críticos fueron acertadamente identificados como tal, lo cual se refleja en una sensibilidad del 100%.

$$\boldsymbol{F1} = \frac{2 * precision * recall}{precision + recall} = \frac{2 * 0.8148 * 1}{0.8148 + 1} = \mathbf{0.90}$$

El rendimiento de esta etapa se resume mediante la métrica F1=0.90.

Evaluación de las decisiones finales

Las decisiones resultantes fueron comparadas con las decisiones óptimas según las condiciones de cada escena y caso. Se han clasificado los errores producidos en dos tipos. Los errores poco relevantes son los que modificarían en mínimo grado el comportamiento del robot, mientras que los errores relevantes representan efectos evidentes y no deseados.

El porcentaje de error en las decisiones resultantes se calculan dividiendo el número de errores encontrados para el número total de casos. De esta manera, el porcentaje de errores poco relevantes es el 16%, mientras que los errores relevantes representan solamente el 10%.

Velocidad de procesamiento en tiempo real

El tiempo empleado para el procesamiento completo varía entre 0.5 y 0.6 segundos por captura, dependiendo de la cantidad de objetos observados en el mapa de profundidades y de la presencia o ausencia de obstáculos. De esta manera, la velocidad promedio de ejecución es de 1,8 FPS (cuadros por segundo).

El principal factor limitante de la velocidad de procesamiento es la capacidad de la unidad central de procesamiento CPU de la tarjeta StereoPi utilizada.

VI. DISCUSIÓN

Los errores presentados en la etapa de estimación de distancia se deben principalmente a que el rango de visibilidad de las cámaras y de generación del mapa de profundidades no alcanza a percibir los extremos de la ruta crítica en algunas de las escenas. Éste es un factor externo que no determina de manera directa el rendimiento del software. Otro motivo de error son las operaciones de morfología matemática aplicadas en el mapa de profundidades.

Los resultados de la etapa de detección de obstáculos son bastante acertados, presentando dificultades únicamente cuando los obstáculos observados son muy pequeños. Sin embargo, esto no representa fallas significativas durante el funcionamiento, ya que los objetos tan pequeños no impiden la movilidad de un robot.

En cuanto a la clasificación de obstáculos, es imprescindible mejorar la exactitud para poder llevar este proyecto a pruebas de campo. El rendimiento de esta etapa estuvo limitado principalmente por la capacidad de procesamiento disponible en la *StereoPi*, la cual restringe el uso de modelos más complejos. Por otro lado, el algoritmo de clasificación presenta un comportamiento conservador, lo cual resulta adecuado para la finalidad del software de salvaguardar la seguridad física de los posibles obstáculos críticos.

Finalmente, las imperfecciones de cada una de las etapas previas se ven reflejadas en los resultados entregados por el módulo de decisiones. Mediante los ajustes necesarios en ellas, las acciones finales sugeridas por esta contribución serán de muy buena calidad.

VII. CONCLUSIONES

Se desarrolló una herramienta de software que, a partir de las imágenes capturadas por una cámara estéreo, es capaz de aproximar el ancho de la ruta que se encuentra por delante, identificar la presencia de obstáculos y clasificarlos según su naturaleza, con el fin de tomar la decisión más conveniente para un robot de navegación autónoma sin poner en peligro la integridad de niños o mascotas que se encuentren en el entorno.

Se implementó un algoritmo de detección de obstáculos mediante la búsqueda exhaustiva en la parte inferior del mapa de profundidades. Este algoritmo encuentra la ruta crítica más cercana y los objetos presentes en ella, en el rango de profundidad entre 0,5m y 2m. Se estima la distancia parcial y total de la ruta crítica en centímetros, en base a su profundidad. El rendimiento de la detección está caracterizado por la medida F1=0.98, y una precisión del 100%, demostrando alta robustez en la localización de obstáculos de tamaño considerable.

Se entrenó un modelo de clasificación mediante aprendizaje profundo, que asigna las capturas de los obstáculos previamente detectados a una de las dos posibles categorías: obstáculos comunes y obstáculos críticos. La arquitectura de la red neuronal incluye 4 capas convolucionales y 1 capa densa oculta. Los resultados obtenidos son conservadores, evitando la generación de falsos negativos, y están representados por la métrica F1=0.90, sensibilidad de 100% y exactitud de 74%.

Una etapa final procesa la información arrojada por los procedimientos previos y arroja la decisión más acertada de acuerdo a estos datos. Se han considerado seis distintas decisiones posibles: continuar ruta, evadir obstáculo y continuar, evadir y proceder con precaución, manipular obstáculo para poder continuar, emitir una alerta sonora y esperar que el objeto crítico permita el paso, o cambiar la ruta. De entre las decisiones calculadas durante las pruebas realizadas, se obtuvo que en menos del 10% de decisiones el comportamiento predicho difiere de manera significativa con las acciones óptimas a ejecutar.

El software desarrollado fue implementado en una tarjeta StereoPi para la evaluación de los resultados obtenidos en 31 casos de prueba distribuidos en 9 distintos escenarios de una casa familiar. La velocidad de procesamiento en el hardware utilizado alcanza un promedio de 2 FPS. A pesar de que el rendimiento de esta herramienta está lejos de ser óptimo, presenta una buena alternativa para aplicaciones de investigación en robótica que

utilicen únicamente una tarjeta basada en RaspberryPi para el procesamiento de información.

Como líneas de trabajo futuro se sugieren las siguientes modificaciones:

- Ampliar el ángulo de visibilidad, a través de cámaras con mayor campo de visión o cámaras adicionales.
- Realizar pruebas de campo en un robot real para observar de manera integral el funcionamiento de este software ante situaciones reales.
- Personalizar el modelo de clasificación para diferenciar objetos de interés específicos o en categorías distintas a las aquí mencionadas
- Ejecutar el procesamiento principal en un ordenado con GPU para optimizar la velocidad, utilizando la StereoPi exclusivamente para la captura y preprocesamiento básico.

REFERENCIAS

[1] N. Ayache, «Recovering the third dimension», en *Artificial Vision for Mobile Robots: Stereo Vision and Multisensory Perception*, P. T. Sander, Ed. Cambridge, Mass.: The Massachusetts Institute of Technology Press, 1991, pp. 7-8.

[2] D. Marr y T. Poggio, «A computational theory of human stereo vision», *Readings Cogn. Sci. A Perspect. from Psychol. Artif. Intell.*, vol. 328, pp. 534-547, 2013, doi: 10.1016/B978-1-4832-1446-7.50046-7.

[3] UW CSE vision faculty, «Stereo and 3D Vision», 2017. [En línea]. Disponible en: https://courses.cs.washington.edu/courses/cse455/09wi/Lects/lect16.pdf. [Accedido: 20-jul-2020].

[4] D. Scharstein, R. Szeliski, y R. Zabih, «A taxonomy and evaluation of dense two-frame stereo correspondence algorithms», *Proc. - IEEE Work. Stereo Multi-Baseline Vision, SMBV 2001*, vol. 47, n.º 1, pp. 131-140, 2001, doi: 10.1109/SMBV.2001.988771.

[5] I. MathWorks, «¿Qué es el reconocimiento de objetos?», *Image Processing and Computer Vision*, 2018. [En línea]. Disponible en: https://la.mathworks.com/solutions/image-video-processing/object-recognition.html. [Accedido: 22-jul-2020].

[6] A. Ramisa, D. Aldavert, S. Vasudevan, R. Toledo, y R. Lopez de Mantaras, «Evaluation of Three Vision Based Object Perception Methods for a Mobile Robot», en *Household Service Robotics*, vol. 68, Zhejiang University Press Co., Ltd, 2015, pp. 303-337.

[7] D. Nistér y H. Stewénius, «Scalable recognition with a vocabulary tree», *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2, pp. 2161-2168, 2006, doi: 10.1109/CVPR.2006.264.

[8] P. Viola y M. Jones, «Rapid Object Detection using a Boosted Cascade of Simple Features», en *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, 2001, pp. 511-518, doi: 10.1109/CVPR.2001.990517.

[9] S. Goyal y P. Benjamin, «Object Recognition Using Deep Neural Networks: A Survey», pp. 1-7, 2014.

[10] A. Krizhevsky, I. Sutskever, y G. E. Hinton, «ImageNet Classification with Deep ConvolutionalNeural Networks», en *Advances in Neural Information Processing Systems 25 (NIPS 2012)*, 2012, pp. 1-9.

[11] O. Russakovsky *et al.*, «ImageNet Large Scale Visual Recognition Challenge», *Int. J. Comput. Vis.*, vol. 115, n.º 3, pp. 211-252, 2015, doi: 10.1007/s11263-015-0816-y.

[12] Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, S. Nasrin, y V. K. Asari, «The History Began from AlexNet: Comprehensive Survey on Deep Learning Approaches», 2017.

[13] J. Redmon, S. Divvala, R. Girshick, y A. Farhadi, «(YOLO) You Only Look Once», en *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.

[14] Y. Guo, M. Bennamoun, F. Sohel, M. Lu, y J. Wan, «3D object recognition in cluttered scenes with local surface features: A survey», *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, n.º 11, pp. 2270-2287, 2014, doi: 10.1109/TPAMI.2014.2316828.

[15] S. Zhi, Y. Liu, X. Li, y Y. Guo, «Toward real-time 3D object recognition: A lightweight volumetric CNN framework using multitask learning», *Comput. Graph.*, vol. 71, pp. 199-207, 2018, doi: 10.1016/j.cag.2017.10.007.

[16] R. C. Michelson, «Autonomous navigation», *AccessScience*, 2000.

[17] Tesla, «Electric Cars, Solar & Clean Energy», *Tesla Webpage*, 2020. [En línea]. Disponible en: https://www.tesla.com/. [Accedido: 28-jul-2020].

[18] Mobile Industrial Robots A/S, «Frontpage», 2020. [En línea]. Disponible en: https://www.mobile-industrial-robots.com/en/. [Accedido: 28-jul-2020].

[19] H. Durrant-Whyte y T. Bailey, «Simultaneous Localization and Mapping: Part I», *IEEE Robotics & Automation Magazine*, vol. 13, n.º 2, pp. 99-110, 2006.

[20] A. J. Davison, «Real-time simultaneous localisation and mapping with a single camera», *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2, pp. 1403-1410, 2003, doi: 10.1109/iccv.2003.1238654.

[21] D. Nistér, O. Naroditsky, y J. Bergen, «Visual odometry», *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 1, 2004, doi: 10.1109/cvpr.2004.1315094.

[22] D. Nistér, «Preemptive RANSAC for live structure and motion estimation», *Mach. Vis. Appl.*, vol. 16, n.º 5, pp. 321-329, 2005, doi: 10.1007/s00138-005-0006-y.

[23] E. Mouragnon, M. Lhuillier, M. Dhome, F. Dekeyser, y P. Sayd, «Real time localization and 3D reconstruction», *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 1, pp. 363-370, 2006, doi: 10.1109/CVPR.2006.236.

[24] H. Lategahn, A. Geiger, y B. Kitt, «Visual SLAM for autonomous ground vehicles», *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 1732-1737, 2011, doi: 10.1109/ICRA.2011.5979711.

[25] G. Zhang, J. H. Lee, J. Lim, y I. H. Suh, «Building a 3-D line-based map using stereo SLAM», *IEEE Trans. Robot.*, vol. 31, n.º 6, pp. 1364-1377, 2015, doi: 10.1109/TRO.2015.2489498.

[26] J. Engel, J. Stückler, y D. Cremers, «Large-scale direct SLAM with stereo cameras», *IEEE Int. Conf. Intell. Robot. Syst.*, vol. 2015-Decem, pp. 1935-1942, 2015, doi: 10.1109/IROS.2015.7353631.

[27] J. Zhang, L. Tai, J. Boedecker, W. Burgard, y M. Liu, «Neural SLAM: Learning to Explore with External Memory», 2017.

[28] M. Bloesch, J. Czarnowski, R. Clark, S. Leutenegger, y A. J. Davison, «CodeSLAM - Learning a Compact, Optimisable Representation for Dense Visual SLAM», *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 2560-2568, 2018, doi: 10.1109/CVPR.2018.00271.

[29] X. Gao y T. Zhang, «Unsupervised learning to detect loops using deep neural networks for visual SLAM system», *Auton. Robots*, vol. 41, n.º 1, pp. 1-18, 2017, doi: 10.1007/s10514-015-9516-2.

[30] S. Milz, G. Arbeiter, C. Witt, B. Abdallah, y S. Yogamani, «Visual SLAM for automated driving: Exploring the applications of deep learning», *IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Work.*, vol. 2018-June, pp. 360-370, 2018, doi: 10.1109/CVPRW.2018.00062.

[31] C. Liu, B. Zheng, C. Wang, Y. Zhao, S. Fu, y H. Li, «CNN-based vision model for obstacle avoidance of mobile robot», *MATEC Web Conf.*, vol. 139, pp. 4-7, 2017, doi: 10.1051/matecconf/201713900007.

[32] H. Yu, R. Hong, X. Huang, y Z. Wang, «Obstacle detection with deep convolutional neural network», en *Proceedings - 6th International Symposium on Computational Intelligence and Design, ISCID 2013*, 2013, vol. 1, n.º 1, pp. 265-268, doi: 10.1109/ISCID.2013.73.

[33] N. Deepika y V. V. Sajith Variyar, «Obstacle classification and detection for vision based navigation for autonomous driving», *2017 Int. Conf. Adv. Comput. Commun. Informatics, ICACCI 2017*, vol. 2017-Janua, pp. 2092-2097, 2017, doi: 10.1109/ICACCI.2017.8126154.

[34] G. Prabhakar, B. Kailath, S. Natarajan, y R. Kumar, «Obstacle detection and classification using deep learning for tracking in high-speed autonomous driving», *TENSYMP 2017 - IEEE Int. Symp. Technol. Smart Cities*, pp. 3-8, 2017, doi: 10.1109/TENCONSpring.2017.8069972.

[35] D. S. Levkovits-Scherer, I. Cruz-Vega, y J. Martinez-Carranza, «Real-Time Monocular Vision-Based UAV Obstacle Detection

and Collision Avoidance in GPS-Denied Outdoor Environments Using CNN MobileNet-SSD», en *Advances in Soft Computing*, 2019, pp. 613-621.

[36] D. Jones, «Picamera», 2013. [En línea]. Disponible en: https://picamera.readthedocs.io/en/release-1.13/. [Accedido: 30-jun-2020].

[37] O. M. Parkhi, A. Vedaldi, A. Zisserman, y C. V. Jawahar, «Cats and dogs», *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 3498-3505, 2012, doi: 10.1109/CVPR.2012.6248092.

[38] T. Karras, S. Laine, y T. Aila, «A style-based generator architecture for generative adversarial networks», *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2019-June, pp. 4396-4405, 2019, doi: 10.1109/CVPR.2019.00453.

[39] G. Griffin, A. Holub, y P. Perona, «The Caltech 256», *Caltech Technical Report*, 2006. [En línea]. Disponible en: http://www.vision.caltech.edu/Image_Datasets/Caltech256/. [Accedido: 03-sep-2020].