Benémerita Universidad Autónoma de Puebla

Facultad de Ciencias de la Electrónica

Identificación de objetos para la estimación de la profundidad mediante un sistema de visión artificial binocular

T E S I S

para obtener el grado de:

LICENCIATURA EN ELECTRÓNICA

Por: Laura Lidia Aragón Mayo

Asesorado por:

Dr. Juan Pablo Flores Flores

Dr. José Eladio Flores Mena

Dr. Antonio Michua Camarillo

Puebla, Pue. 24 de febrero de 2024

Título: Identificación de objetos para la estimación de la profundidad mediante un sistema de visión artificial binocular

Estudiante: Laura Lidia Aragón Mayo

CÓMITE

Dr. Richard Torrealba Meléndez
Presidente

Dr. José Eladio Flores Mena
Asesor

Dr. Enrique Morales Rodríguez
Secretario

Dr. Antonio Michua Camarillo
Asesor

Dr. Plácido Zaca Morán
Vocal

Dr. Juan Pablo Flores Flores
Asesor

Dedicatorias

"Una mente que se abre a una nueva idea nunca volverá a su tamaño original"

-Albert Einstein(1879-1955)

A mi familia y amigos

Agradecimientos

Quiero agradecer a mis asesores el Dr. José Eladio Flores Mena y el Dr. Antonio Michua Camarillo por brindarme la oportunidad de conocer y trabajar con grandes profesionales, aprender de sus experiencias y desarrollar habilidades que me servirán toda la vida.

Desde lo personal quiero agradecer al Dr. Juan Pablo Flores Flores; mi mentor, cuya paciencia fue puesta a prueba en incontables ocasiones, además por su dedicación y enseñanza. Todo el trabajo realizado fue posible gracias a su apoyo incondicional.

Estoy en deuda con mis padres, Laura Mayo Cuevas y Fortino Aragón Sumano, que me han dado todo lo que necesité.

También quiero reconocer el incansable apoyo de mi abuelita Magdalena Cuevas López, cuyo ejemplo de cáracter y tenacidad ha sido un pilar en mi formación.

A mi hermano Jesús Alberto Aragón Mayo por su compañia, palabras de aliento y risas.

A mis amigos, en especial a mi mejor amiga Rubí por estar en los buenos y malos momentos.

A mis profesores, a quienes siempre tendré en mi memoria.

Índice general

Índice de figuras

Resumen

El presente trabajo de tesis se halla en el contexto de la *visión artificial* o *visión por computadora*, una de las disciplinas con mayor impacto en la sociedad moderna, así como en básicamente todas las disciplinas científicas y tecnológicas modernas. En particular, aquí se explora la *visión binocular por computadora*. Se implementó un arreglo binocular conformado por dos cámaras USB para estimar la profundidad o distancia a la cual se encuentra algún objeto de interés del arreglo mismo. Esto se consigue calculando la disparidad, es decir, el desplazamiento horizontal de un punto 3D proyectado en una imagen con respecto a su proyección en la otra imagen. Dado que se desea estimar la distancia de objetos específicos, es necesario primero determinar su ubicación en las imágenes. Por tal motivo, se exploran algunas de las diferentes aproximaciones para la detección de objetos. A lo largo de este trabajo se presentan nociones básicas de la visión binocular por computadora, se comenta la conformación del arreglo de cámaras, mismo que es implementado con ayuda del *Sistema Operativo de Robots* (ROS) y se presentan los resultados obtenidos.

Introducción

En la actualidad el mundo está experimentando la llamada revolución de la *Inteligencia Artificial* (IA). Ya desde 2016, el Foro Económico Mundial preveía grandes cambios en la sociedad, ligados principalmente a la forma de hacer negocios, la organización gubernamental y la vida cotidiana en general [1]. Dichos cambios son hoy en día una realidad y permiten una constante innovación en prácticamente todas las disciplinas científicas y tecnológicas, tales como la robótica, la mercadotecnia, la medicina, la ciberseguridad, etc. [2–4]. Una disciplina que se beneficia y al mismo tiempo auxilia a la IA es la *visión por computadora* o *visión artificial*. Si bien esta disciplina no es reciente, su importancia va en aumento día con día, así como su número de aplicaciones tanto en el ámbito académico como industrial.

Actualmente, la visión artificial binocular se utilizan en diferentes campos, como el entretenimiento, la navegación, el procesamiento y análisis de imágenes, la tele-cirugía, los simuladores de vuelo, entre otros. Algunos trabajos recientes muy interesantes son: donde los autores presentan un sistema de visión binocular para el guiado de un robot quirúrgico mediante la identificación de los dedos del cirujano dentro de la cavidad abdominal del paciente [5]. Por otro lado, en [6] se implementa un sistema de visión en un robot con la capacidad de identificar y evadir objetos, cuyo propósito es la asistencia de personas con discapacidad motora. A su vez, J. Jener en [7] se presenta un sistema robotizado controlado por señales cerebrales mediante una interfaz cerebro-computadora en el que se implementa un sistema de visión artificial. Por otro lado, la visión binocular también es ampliamente utilizada en los procesos de manufactura y control de calidad. Tales procesos pertenecen a una gran diversidad de industrias y cubren desde objetos de uso común hasta productos altamente especializados, como microprocesadores, circuitos integrados, etc. [8].

Evidentemente, la visión artificial binocular es hoy en día una de las tecnologías con mayor presencia y desarrollo en el mundo. No obstante, en nuestro país, el rezago de esta área es evidente, existiendo muy pocos grupos de investigación dedicados a este campo. Sumado a esto, existe una escasez y al mismo tiempo una gran demanda de profesionistas especializados o familiarizados con

estas tecnologías [9]. Por lo tanto, con este trabajo de tesis, adicionalmente se busca contribuir con una mayor presencia de este tipo de trabajos en la academia, particularmente en la Benemérita Universidad Autónoma de Puebla, así como ampliar el panorama profesional del autor de esta tesis, con respecto a las técnicas de detección de objetos y el uso de Redes Neuronales, esto es, introducirse a un tópico altamente demandado por la industria.

¿Por qué visión artificial binocular?

Una de las capacidades más útiles e interesantes que ofrece la visión es la de percibir profundidades, y así como sucede en los sistemas de visión biológicos, para que una maquina u ordenador pueda generar información de profundidad de su entorno, es necesario de un arreglo de visión que proporcione dos o mas imágenes de una misma escena pero ligeramente diferentes entre sí.

Bajo esta premisa, se propone el presente trabajo de tesis, cuyo objetivo principal o general es la implementación de un arreglo de visión artificial binocular para la estimación de profundidad de distintos objetos de interés. No obstante, dado que para determinar la profundidad o distancia a la que se halla un objeto particular es necesario antes detectar dicho objeto, el objetivo general se puede replantear como:

- **Objetivo General:** Implementación de un arreglo de visión artificial binocular para la detección de objetos y posterior estimación de la distancia a la que se hallan del arreglo de visión.

Objetivos particulares:

- Revisar y comprender acerca de los distintos algoritmos de detección de objetos. Se propone estudiar los mas comunes en la actualidad, estos son: detección por color, haarcascades y Redes Neuronales Convolucionales.
- Revisar y comprender la técnica de geometría proyectiva, responsable de la estimación de la profundidad.
- Estudiar los conceptos básicos del Sistema Operativo Robótico (ROS) para la posterior integración de un arreglo de visión artificial binocular con un ordenador capaz de procesar la información recolectada por el arreglo.
- Experimentar con el arreglo de visión binocular. Particularmente se busca comparar los resultados obtenidos al implementar cada uno de los algoritmos de detección de objetos que se propone estudiar.

Este documento de tesis esta estructurado de la siguiente manera: En el primer capítulo se proporciona una breve descripción de los conceptos necesarios para el desarrollo de este manuscrito; incluyendo conceptos de visión binocular en seres vivos así como conceptos del proceso de visión artificial, por lo que se habla de la formación de la imagen y el tratamiento de la misma. Por otro lado, el capítulo también hace mención a las técnicas de detección de objetos utilizadas aquí. A continuación, el segundo capitulo incluye la descripción de la construcción del sistema de visión binocular, así como el metódo de implementación, es decir, las herramientas utilizadas para su conformación. Después, en el capítulo tres se presentan los resultados de estimación de profundidad obtenidos tomando en cuenta cada una de las técnicas de detección de objetos. Adicionalmente se lleva a cabo una breve discusión al respecto. Por último, en la sección de conclusiones presentamos, las observaciones más relevantes de este trabajo de tesis así como los posibles desafíos y/o trabajos a futuro que podrían desprenderse de aquí.

Capítulo 1

Fundamentos

En este capítulo se ofrece una breve revisión de los conceptos vinculados con la visión binocular, porque en general los sistemas de visión artificial tienen un funcionamiento que es análogo al sistema de visión del ser humano. En este trabajo se hace particularmente enfásis en la capacidad del ser humano de percibir profundidades; como uno de los diferentes mecanismos del sistema de visión. Se hace también una revisión de los conceptos teóricos relacionados con la visión por computadora o visión artificial, como lo es la etapa de adquisición de una señal para la generación de la imagen digital. Dado que se va a trabajar con esta información visual, se habla de su representación digital y se describen de forma general los algoritmos para el tratamiento de la misma.

1.1. Visión binocular

Antes de abordar el tema de *visión artificial*, es importante comprender que es *visión* y en particular, que es *visión binocular*. Se sabe que de entre las distintas capacidades sensoriales que poseen los seres vivos, la visión es de las mas importantes, y en muchos casos, la mas importante. Esto se debe a que la información visual es sumamente útil para diferenciar entre el día y la noche, percibir distancias, detectar potenciales amenazas en el entorno, orientarse y en general, para navegar.

Si bien no existe un órgano visual único, así como tampoco una disposición única de estos, entre los organismos más complejos suele ser una constante la existencia de dos órganos visuales dispuestos horizontalmente, es decir, a la misma altura de la cabeza. Este es el caso de los seres humanos, y tal configuración visual es conocida como *arreglo binocular*.

Una de las capacidades mas útiles que ofrece el sentido de visión es la de percibir profundidades. Si bien percibir las distancias horizontales o verticales con un solo órgano visual es relativamente sencillo, detectar profundidades puede ser desafiante. Esto se debe a que para reconstruir escenas tridimensionales, el cerebro necesita de al menos dos imágenes bidimensionales ligeramente diferentes entre sí. Esta capacidad se conoce como *estereopsis*, la cual solo es posible con un sistema de visión con dos o más órganos visuales, tal como un sistema de visión binocular.

El fenómeno de estereopsis consiste en la recaudación simultánea de imágenes por parte de ambos ojos. Tales imágenes corresponden a una misma escena pero presentan ciertas diferencias entre sí debido precisamente a la separación de los ojos. Como consecuencia, se produce un aparente desplazamiento de una imagen con respecto a la otra, desplazamiento que es conocido como *disparidad retinal* y es interpretado por el cerebro como una medida de profundidad [10].

El fenómeno de la estereopsis, fue inicialmente estudiado por Euclides durante el siglo III a.C., quien describió como el ojo izquierdo y el derecho ven una esfera de forma distinta. Posteriormente, en el siglo II d.C., el físico Claudi Galleni, notó que al abrir primero el ojo izquierdo y acto seguido el derecho, cada uno veía diferentes porciones del fondo detrás de una columna. Por otro lado, Leonardo Da Vinci anticipó que es imposible que los objetos pintados tengan el mismo relieve que la realidad, a menos que esta se vea con un solo ojo. Estos conceptos dieron lugar a posteriores investigaciones que condujeron a la primera aplicación real del fenómeno de estereopsis con el llamado "espejo reflector estereoscópico" (1830). Este fue el primer dispositivo capaz de generar un efecto tridimensional a partir de imágenes bidimensionales y fue diseñado por el físico Charles Wheatstone [11].

Como regla general, para que un sistema de visión binocular sea capaz de reconstruir una escena tridimensional, necesita satisfacer las siguientes condiciones [12]:

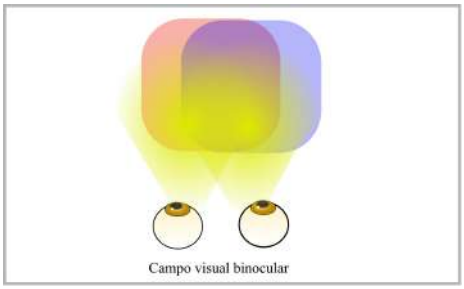


Figura 1.1: Intersección de los campos de visión monocular.

1) Los dos *campos visuales monoculares* deben superponerse en una región suficientemente amplia para obtener un *campo visual binocular* extenso. El cam-

po visual monocular es el espacio visible por un órgano visual. Por otro lado, el campo visual binocular es la región formada por la intersección de los dos campos monoculares (Figura 1.1 Campo visual monocular del órgano visual izquierdo (rojo), campo monocular del órgano derecho (azul) y campo visual binocular definido por la intersección de los primeros (morado).

2) Los órganos visuales deben moverse de forma coordinada para que los ejes visuales se crucen sobre un mismo punto de fijación. Esto permite obtener imágenes nítidas y superpuestas. Cuando no hay un movimiento coordinado, el resultado puede ser una falta de superposición de imágenes, lo que provoca visión doble.

3) La información recibida por cada órgano visual debe transmitirse de tal manera que se conserve la correspondencia temporal de las imágenes. Por lo tanto, no debe haber interferencia en la conexión entre los órganos visuales y el cerebro.

4) El cerebro debe tener la capacidad de fusionar las imágenes obteniendo una representación única. Es decir, debe ser capaz de crear una sola imagen a partir de la superposición o fusión de las imágenes de cada órgano visual.

Los seres humanos y la gran mayoría de animales, particularmente los depredadores, satisfacemos estas condiciones, por lo cual somos capaces de percibir la profundidad de nuestro entorno, o en otras palabras, podemos percibir visualmente la distancia a la que se encuentran los objetos a nuestro alrededor.

1.2. Visión artificial

La visión artificial es una disciplina científica que carece de una definición universalmente aceptada. Por ejemplo, la compañía tecnológica IBM la define como el campo de la inteligencia artificial que permite a una computadora extraer información relevante a partir de imágenes digitales, vídeos y cualquier otro tipo de entrada visual, con la finalidad de tomar decisiones y/o hacer recomendaciones. Por otro lado, Alegre *et al.* [13] la definen como la disciplina encargada de deducir automáticamente la estructura y propiedades de nuestro mundo tridimensional a partir de una o varias imágenes bidimensionales.

Entonces, podemos entender la visión artificial o visión por computadora como la disciplina científica encargada de dotar a una máquina con la capacidad de "ver" su entorno y extraer información útil del mismo. Es decir, es el conjunto de técnicas destinadas a la adquisición, análisis y procesamiento de información visual mediante dispositivos electrónicos, con la finalidad de dotar a una maquina con información adicional que le permita realizar tareas más complejas.

Durante las últimas dos décadas, el incremento de las capacidades de procesamiento y almacenamiento de datos de los ordenadores ha permitido que la visión por computadora experimente un importante desarrollo. Sumado a esto, la implementación de los llamados *algoritmos inteligentes* ha acelerado aún más este proceso, dando como resultado una inmensa diversificación de las aplicaciones [14]. Tales aplicaciones van desde la manufactura asistida, pasando por los vehículos autónomos y la agricultura, hasta el diagnóstico y tratamiento de enfermedades [15].

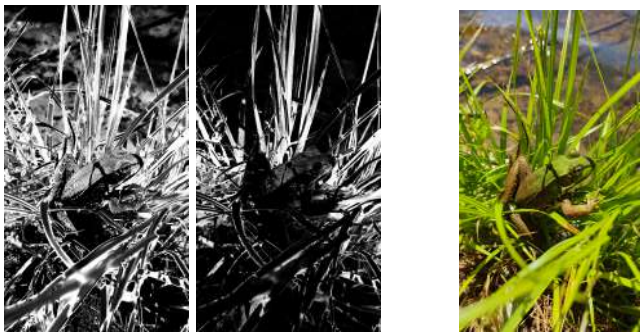
Actualmente, la visión por computadora es utilizada principalmente para desempeñar las siguientes tareas (Figura 1.2):

- **Clasificación de imágenes**: Consiste en determinar si una imagen contiene algún objeto que pertenezca a una categoría específica. Por ejemplo, determinar si en la imagen hay un auto, sin importar su modelo, color, tamaño, etc.
- **Detección o localización**: Busca determinar la ubicación de un objeto dentro de la imagen, es decir, establecer las coordenadas de dicho objeto dentro de la imagen. Esto se puede hacer para uno o varios objetos en la imagen.
- **Identificación de objetos**: Esta es una tarea muy similar a la anterior, excepto que se hace para objetos únicos. Por ejemplo, identificar el rostro de una persona o el logotipo de una compañía en particular
- **Segmentación de imágenes**: Consiste en determinar que región o regiones de la imagen pertenecen a un mismo objeto.
- **Seguimiento de objetos**: Esta tarea ubica un mismo objeto a lo largo de una serie temporal de imágenes, es decir, localiza un objeto en movimiento.
- **Estimación de la profundidad**: Esta consiste en determinar la distancia a la que se halla uno o varios objetos del sistema de visión que los observa.

Cada una de estas tareas es de gran relevancia y se encuentra en continuo desarrollo. Más auń, las mismas suelen implementarse de forma combinada para poder llevar a cabo actividades mas complejas, como navegación o supervisión de procesos.

1.3. Cámaras e imágenes digitales

Así como los ojos en el sistema visual del ser humano, en un sistema de visión artificial, son las cámaras digitales los dispositivos encargados de capturar la luz



(a) (b) (c)

(d)

Figura 1.2: Tareas de visión artificial: Segmentación de imagen basada en sus regiones de color: a) amarillo, b) verde y c) original. d) Detección o localización de rostros.

que reflejan los objetos que son iluminados por una o mas fuentes de radiación para posteriormente generar una imagen (Figura 1.3). En una cámara digital, la luz recibida se hace incidir sobre un *sensor fotosensible*, para posteriormente generar señales digitales que llevan información de luminosidad y color. Estos sensores fotosensibles, comúnmente conocidos como sensores ópticos, presentan una estructura reticular, es decir, están conformados por miles o millones de *celdas fotosensibles* dispuestas una al lado de otra sobre una superficie plana comúnmente con forma de paralelogramo de ángulos rectos (Figura 1.4). Cada una de estas celdas fotosensibles genera una señal digital que depende de la cantidad de luz que reciben.

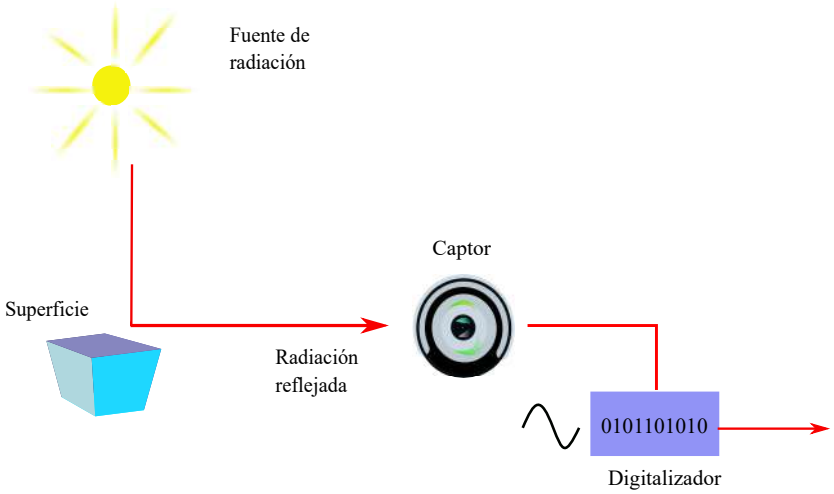


Figura 1.3: Para la formación de una imagen digital, un sistema óptico artificial captura la luz reflejada por la superficie de los objetos en la escena. Dicha información es procesada para generar señales digitales, las cuales pueden ser posteriormente interpretadas por un ordenador como arreglos numéricos.

Otro elemento de gran relevancia en una cámara digital, es el *objetivo*. El objetivo consiste en un conjunto de lentes cuya función es dirigir la luz reflejada por la escena hacia el sensor fotosensible. El objetivo es responsable de la *exposición* y del *enfoque* de la cámara digital, es decir, es capaz de regular la cantidad de luz que incide sobre el sensor fotosensible, así como controlar el contraste y la nitidez de la imagen digital resultante.

Dado que la información capturada por el sensor fotosensible es analógica, para que un ordenador pueda interpretarla, la misma debe ser codificada en una señal digital. La forma de conseguir esto es traduciendo dicha información a un conjunto de bits de longitud determinada, concretamente en bytes (1 byte = 8 bits). Los bytes están en un formato binario que reconoce dos estados, activo (1) e inactivo (0), de tal manera que los bytes son cadenas numéricas conformadas por ceros y unos, e.g.: 00101110. Si bien información en forma de

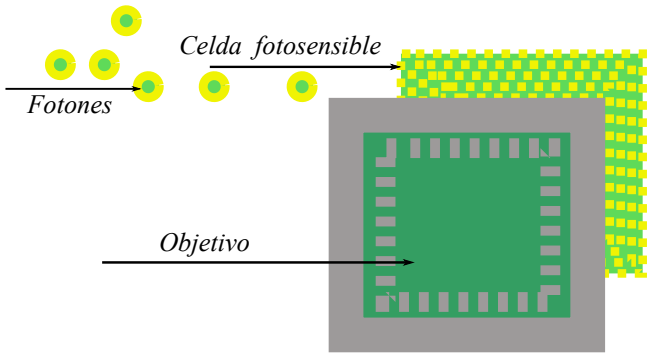


Figura 1.4: Sensor fotosensible: Gracias al *efecto fotoeléctrico*, cada celda del sensor genera una señal eléctrica cuya intensidad depende de la cantidad de luz que recibe. La cantidad de luz a su vez es dirigida y puede ser regulada por el objetivo de la cámara.

bytes es fácilmente interpretable por un ordenador, para los seres humanos esto resulta bastante complicado. Por tal motivo, los bytes de información son nuevamente traducidos e interpretados como un arreglo numérico, e.g., una matriz de dimensiones $n \times m$.

En dicho arreglo numérico, se tiene que cada elemento del mismo representa un *píxel*. Se entiende por píxel a la mínima región de color homogéneo, tal que un conjunto de píxeles conforman una imagen digital. Si bien no es totalmente correcto, en la gran mayoría de literatura, suele asociarse el número de celdas fotosensibles con el número de píxeles y por lo tanto, con la *resolución* de la imagen que genera una cámara digital. En otras palabras, a mayor cantidad de píxeles, mayor será la nitidez o cantidad de detalle de la imagen resultante, tal y como se puede observar en la Figura 1.5.

1.3.1. Espacios de color

Como se mencionó, una imagen digital esta conformada por un conjunto de píxeles, los cuales son regiones homogéneas de color. Dicho conjunto es representado en un ordenador como un arreglo numérico, cuyos componentes pueden ser a su vez un único valor numérico o un conjunto de números. Lo anterior dependerá principalmente del llamado *espacio de color* de la imagen.

Un espacio de color es definido por un conjunto de fórmulas matemáticas que permiten describir colores en función de diversos criterios, tales como tono, brillo y saturación. Los espacios de color mas conocidos y fáciles de comprender consisten en la representación visual de cualquier color a partir de un conjunto base de colores. En otras palabras, existen ciertos espacios de color que con-

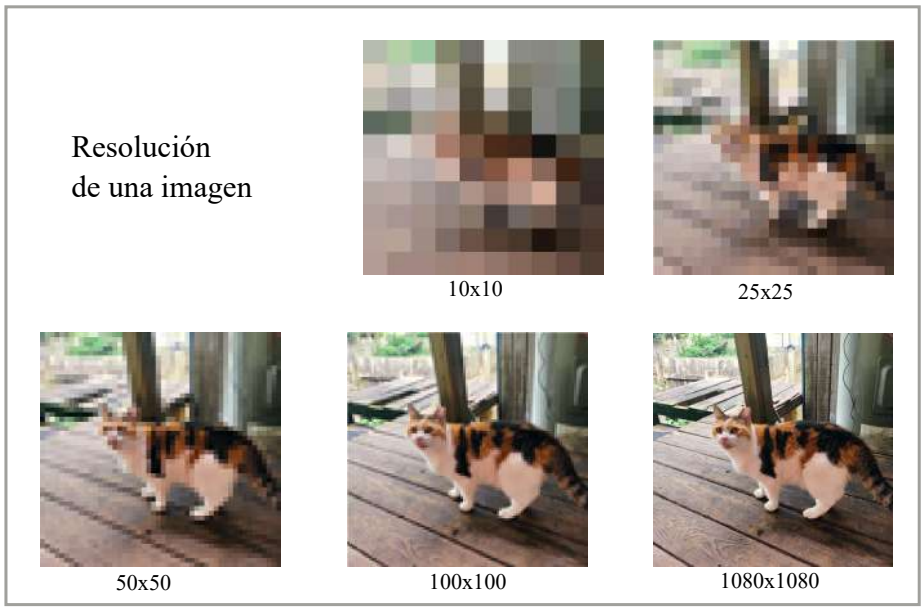


Figura 1.5: La resolución de una imagen esta dada por el numero de píxeles que la conforman. Así, si la imagen consta de 1600×1200 píxeles, se dice que tiene una resolucion de 1 920 000 píxeles, es decir 1.92 Mega píxeles.

sisten en representar una gran variedad de colores mediante la mezcla o suma aritmética de un conjunto reducido de colores.

Las características comúnmente utilizadas para distinguir un color de otro son: brillo, matiz y saturación.

- **Brillo**: Característica que incorpora la noción cromática de intensidad
- **Matiz**: Es un atributo asociado con la longitud de onda dominante en la mezcla de longitudes de onda de la luz. Esto es, se puede interpretar como el color percibido por el observador, cuando se dice que un objeto es rojo, naranja o amarillo, realmente se esta especificando el matiz.
- **Saturación**: Se refiere a la pureza relativa o a la cantidad de luz blanca mezclada con un matiz, el grado de saturación es inversamente proporcional a la cantidad de luz blanca añadida.

El matiz y la saturación, cuando se toman conjuntamente, se denominan cromaticidad. Por lo tanto, se dice que se puede caracterizar un color por su brillo y cromaticidad. Estas características son particularmente importantes para espacios de color orientados hacia el hardware (monitores, impresoras en color) y/o hacia aplicaciones de tratamiento de imágenes.

Los espacios de color más comúnmente utilizados en función del hardware son:

- **RGB** Red, Green, Blue (Rojo, Verde, Azul): Para monitores en color y una amplia gama de video cámaras.
- **CMYK** Cyan, Magenta, Yellow, Black (Cian, Magenta, Amarillo, Negro): Principalmente para impresoras en color.

Respecto a las aplicaciones, básicamente se debe considerar el procesamiento de imágenes. Para esto, se utilizan principalmente:

- **RGB**
- **HSV** (Hue-Matiz, Saturation-Saturación, Value-Valor).
- **HSI** (Hue-Matiz, Saturation-Saturación, Intensity-Intensidad).
- **Escala de grises**.

El espacio *RGB* es un método basado en la adición de diferentes proporciones de cada uno de de los colores primarios de la luz: rojo, verde y azul; la variación de las cantidades de estos colores primarios permite obtener diferentes colores x por mezcla, o suma aritmética [13].

$$x = r + g + b \tag{1.1}$$

Por ejemplo, en la Figura 1.6 es posible observar la descomposición de una imagen RGB en sus distintos canales, es decir, en los tres colores primarios de la luz. La popularidad del espacio RGB va mas allá de su utilidad en función del harware o aplicación, pues se trata además del espacio de color más intuitivo y por lo tanto, más fácil de entender.

El espacio *CMYK* también consiste en la adicion de colores base. Permite representar una gama de colores más amplia, y tiene una mejor adaptación a los medios industriales. Los colores Cyan, Magenta, Yellow y Blac*k* (cian, magenta, amarillo y negro) son los colores secundarios de la luz o alternativamente los primarios de los pigmentos. El modelo CMYK se basa en la absorción de la luz. El color que presenta un objeto corresponde a la parte de la luz que incide sobre este y que no es absorbida por el objeto.

Si bien los colores basados en adición son relativamente sencillos de entender, poseen una gran desventaja: su alta sensibilidad a los cambios de iluminación de la escena. Por tal motivo, también se utilizan otros espacios que caracterizan un color en base a criterios distintos.

Figura 1.6: Separación de una imagen en los distintos canales del espacio de color RGB.

Uno de estos es el espacio *HSI* (Hue, Saturation, Intensity), el cual también suele ser llamado *HSB* por Brightness. Este espacio de color, además de su baja sensibilidad a los cambios de iluminación, es capaz de representar con mayor fidelidad ciertos colores que se encuentran en la naturaleza. Por ejemplo, el color púrpura puede representarse de mejor manera al segmentar colores basándose en el tono de estos, en este caso una luz verde y otra roja, cada una con su tono característico (hue).

El modelo de color HSI es bastante utilizado debido a dos hechos fundamentales: 1) la componente de intensidad I, se puede separar de la información del color en la imagen, y 2) las componentes de matiz y saturación están íntimamente relacionadas con el modo en que los humanos percibimos el color. Estas características hacen del modelo HSI una herramienta ideal para desarrollar algoritmos de procesamiento de imágenes basados en alguna de las sensaciones de color del sistema visual humano.

El espacio de color HSV es bastante similar al HSI, por lo cual también es ampliamente utilizado para el procesamiento de imágenes, no obstante, es un espacio de color un tanto más popular. Debido a sus características, ambos espacios de color son ideales para la segmentación por color, es decir, permiten aislar regiones de una imagen de un color específico.

Cuando se utiliza alguno de los espacio de color anteriores, es posible intuir que los componentes del arreglo numérico que representa a la imagen son en realidad un conjunto de valores, e.g., en RGB, cada componente de arreglo numérico es en realidad una tripleta de valores (un valor para especificar la intensidad de cada color primario).

No obstante, existen otros espacios de color capaces de representar cada píxel de la imagen con un único valor numérico. Este es el caso de la llamada *escala de grises*, en la cual cada píxel posee un valor equivalente a una graduación de gris, es decir, las imágenes representadas de este tipo están compuestas de sombras de grises.Dada su capacidad de representar una imagen con pocos valores

(a) (b) (c)

Figura 1.7: Imagen en diferentes espacion de color: a) escala de grises, b) CYMK y c) HSV.

numéricos, la escala de grises suele ser muy utilizada para ahorrar espacio de almacenamiento y para reducir la capacidad de procesamiento necesaria.

Finalmente, es importante mencionar que el arreglo numérico de toda imagen digital puede ser manipulado para representar dicha imagen en cada uno de los espacios de color anteriormente mencionados [16], e.g., en la Figura 1.7 es posible observar la misma imagen en escala de grises, en CYMK y en HSV. Cada una de dichas imágenes es obtenida mediante la manipulación matemática del arreglo numérico original (RGB).

1.4. Algoritmos para la detección de objetos

Posterior a la obtención de imágenes digitales en un espacio de color conveniente, el siguiente paso es el procesamiento de dichas imágenes. En particular, dado el objetivo de este trabajo de tesis, dicho procesamiento consiste en:

1) **Detectar objetos** de interés en la imagen, esto es, determinar las coordenadas de dichos objetos en ambas imágenes.

2) **Estimar la profundidad** mediante la técnica de geometría proyectiva a partir de las coordenadas previamente obtenidas.

La tarea de detección de objetos consiste en determinar las coordenadas dentro de la imagen de uno o varios objetos que satisfagan ciertos criterios. Tales criterios pueden ser tan simples como el color del objeto, hasta cuestiones mas complejas como la forma, bordes y/o patrones.

Los diversos algoritmos para la detección de objetos en un imagen se han ido diversificando con el tiempo gracias a que los ordenadores cada vez son mas

capaces de sostener el coste computacional necesario. En este trabajo de tesis, nos concentraremos en tres algoritmos para la detección de objetos, estos son: detección por color, Haarcascades y Redes Neuronales Convolucionales.

1.4.1. Detección por color

La segmentación del color en una imagen es una tarea de gran relevancia para una gran cantidad de sistemas de análisis de imágenes dada la importancia de sus resultados. La segmentación de una imagen puede considerarse como la partición de esta en un conjunto de regiones de acuerdo a una serie de criterios.

De manera general, durante el análisis de una región de la imagen se debe considerar un conjunto de descriptores basados en la intensidad y en las propiedades espaciales. Por ejemplo, la proximidad geométrica juega un papel vital en la segmentación. Los píxeles que se encuentran en una misma vecindad tienden a tener propiedades estadísticas muy similares y por lo tanto, es mas probable que pertenezcan a una misma región. Los algoritmos de segmentación de imágenes deben tomar en consideración tanto la homogeneidad de los valores de intensidades como la proximidad que exista entre los píxeles para producir imágenes con regiones conectadas.

(a) (b)

Figura 1.8: Ejemplo de segmentación de color: a) imagen original y b) imagen segmentada por similitud de los niveles de color.

Existe una gran variedad de criterios o métodos de segmentación de imagen, no obstante los métodos clásicos son [17]:

- **Métodos basados en el umbral del histograma de la imagen**: Se trata de métodos probabilísticos que se valen del histograma de la imagen para definir un umbral que permita ser utilizado como criterio de comparación para el agrupamiento de los píxeles.

- **Métodos basados en la detección de discontinuidades**: Estos métodos segmentan la imagen en base a cambios bruscos en los píxeles de la imagen, en otras palabras, definen regiones en base a la similitud de los píxeles con sus respectivas vecindades .
- **Métodos basados en la propiedad de similitud de los niveles de color**: En este caso se usan criterios de homogeneidad de color para la agrupación de los píxeles, es decir, se definen regiones a partir de la similitud del color de los píxeles con sus respectivas vecindades. En otras palabras, se agrupan píxeles con valores numéricos cercanos.
- **Métodos heurísticos de segmentación**: Estos métodos basan su operación en el conocimiento previo de la imagen a segmentar y en la experiencia del observador. Por tal motivo, también suelen ser llamados métodos supervisados de segmentación.

A partir de estas descripciones, es posible intuir que la detección de objetos por color es en realidad un algoritmo de segmentación de imagen basado en la similitud de los niveles color, es decir, se trata de un algoritmo cuyo propósito es aislar los píxeles de una imagen que pertenecen a un color o rango de colores específicos, tal y como se muestra en la Figura 1.8.

1.4.2. Haarcascades

La detección de objetos utilizando haarcascades es un método efectivo basado en el aprendizaje automático donde una función en cascada se entrena a partir de muchas imágenes positivas y negativas, es decir, el ordenador recibe información de imágenes previamente clasificadas. Este proceso de "aprendizaje" permite al ordenador analizar imágenes no vistas con anterioridad y poder detectar en ellas objetos similares a los vistos anteriormente.

Se dice que se emplea una función en cascada dado que la misma está compuesta por una serie de clasificadores cuya función es analizar progresivamente la imagen. A su vez, estos clasificadores están basados en características conocidas como Haar [18].

Dado que una imagen se interpreta como un arreglo numérico, e.g., una matriz, es posible considerar submatrices de la misma imagen y realizar operaciones con estas. En particular, haarcascades multiplica dichas submatrices por una serie matrices definidas por el usuario. Tales matrices de menor dimensión son los clasificadores de la función en cascada y reciben el nombre de núcleos o kernels (1.9). Los kernels desempeñan el rol de un filtro, esto es, resaltan determinadas propiedades de la imagen. La idea general del algoritmo de haarcascades es tomar submatrices de la imagen y multiplicarlas por uno o varios kernels. Este proceso comienza en la parte superior de la imagen y se desliza de izquierda a

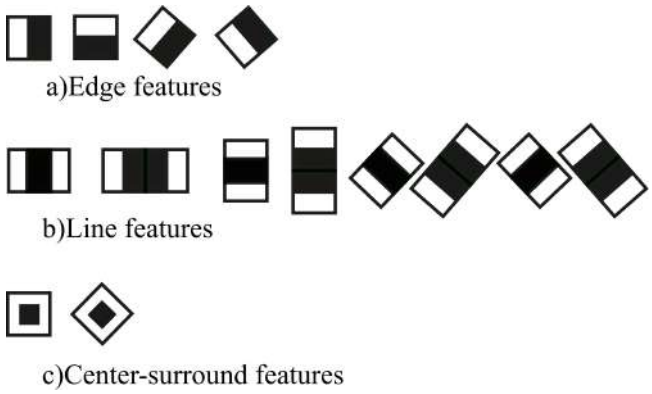


Figura 1.9: Los kernels basados en características o Haar son arreglos numéricos de distinta dimensión, los cuales visualmente pueden ser interpretados por los iconos mostrados aquí. Dada su composición, cada kernel resaltara determinadas características o propiedades de la imagen, tales como bordes, lineas rectas o verticales, etc.

derecha y de arriba a abajo hasta que el kernel o kernels actúan sobre todos las submatrices de la imagen. Posteriormente, se repite este proceso pero considerando una región de la imagen de mayor tamaño, y así sucesivamente hasta que la región a considerar es la imagen completa. El resultado de llevar estas operaciones entre submatrices es la extracción de características de la imagen, tales como bordes, regiones claras y oscuras, patrones geométricos, etc., tal y como se muestra en la Figura 1.10, donde los bordes de la imagen son resaltados gracias a un determinado kernel que actúa sobre la imagen original.

(a) (b)

Figura 1.10: Los kernels toman una imagen de entrada b) y generan otra de salida a) donde determinadas características son resaltadas, tales como los bordes en una imagen.

Finalmente, las características extraídas de una imagen son comparadas con aquellas "aprendidas" de las imágenes previamente vistas. En otras palabras, haarcascades busca identificar patrones previamente aprendidos en las distintas subregiones o submatrices de la imagen. Por otro lado, como se mencionó, Haarcascades considera subregiones de distinto tamaño. Esto se hace para incrementar la precisión del algoritmo. Así, si bien un tamaño inicial pequeño permite minimizar el error del algoritmo e incrementar su sensibilidad, también incrementa su coste computacional.

Las grandes desventajas de usar haarcascades es que tienden a detectar falsos positivos y dado que es necesario entrenar una función de cascada para una clase particular de objetos, la diversidad de objetos que pueden detectar estos algoritmos es muy poca. Sin embargo siguen siendo una parte importante en el tratamiento de imágenes y visión artificial dado su relativo bajo costo computacional. Por tal motivo, esta técnica es aún ampliamente utilizada en dispositivos con recursos limitados.

1.4.3. Redes Neuronales Convolucionales

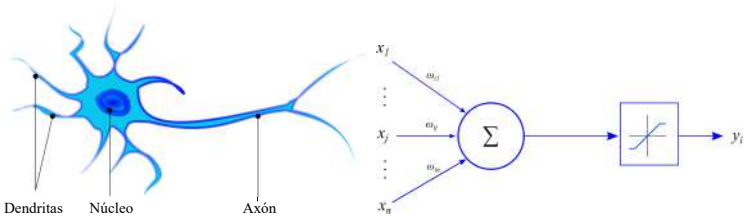
Redes neuronales artificiales

La neurona biológica es una célula especializada en procesar información. Está compuesta por un cuerpo y dos tipos de ramificaciones a través de las cuales recibe y transmite señales eléctricas a otras neuronas. Dichas señales son generadas dentro del cuerpo de la misma célula y contienen información esencial para controlar y regular las funciones del cuerpo [19].

Inspirado en las neuronas biológicas, en 1957 Frank Rosenblatt propuso la idea de una *neurona artificial* [19]. Una neurona artificial, es un elemento diseñado para procesar una suma ponderada de sus valores de entrada, y cuya salida es transmitida a otras neuronas para ser procesada nuevamente. En la Figura 1.11 es posible observar la similitud conceptual entre una neurona biológica y una neurona artificial, esto es, una neurona artificial consta de ramificaciones de entrada y salida, así como una neurona biológica posee dendritas (ramificaciones de entrada) y un axón (ramificación de salida).

Así como una neurona artificial pretende emular una neurona biológica, un conjunto de neuronas artificiales interconectadas pretenden emular de manera muy sencilla al cerebro. A este conjunto de neuronas artificiales interconectadas se les conoce como *red neuronal artificial*, comúnmente abreviada como ANN o NN por sus siglas en nglés (Artificial Neural Network).

La estructura clásica de una red neuronal consta de mayor o menor cantidad de neuronas en función de su complejidad. Dichas neuronas están interconecta-



(a) (b)

Figura 1.11: Similitud conceptual entre una neurona biológica (a) y una neurona artificial (b).

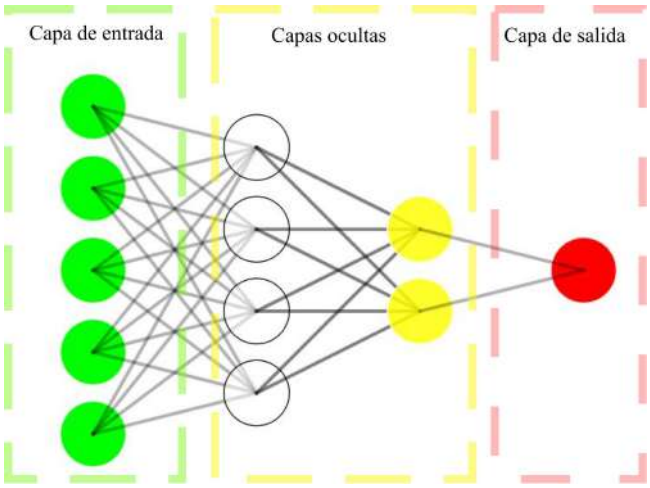


Figura 1.12: Estructura clásica de una red neuronal artificial. Si bien existen muchos tipos de capas, en general es posible clasificarlas en: capa de entrada, capas ocultas y capa de salida.

das entre sí y organizadas en capas o niveles (Figura 1.12). Así, las neuronas de toda red neuronal artificial pertenecen a alguna de las siguientes clases:

- **Neuronas de entrada**: Reciben las señales exteriores. Integran la primera capa de la red.
- **Neuronas ocultas**: Producen resultados intermedios.
- **Neuronas de salida**: Su salida es observable desde el exterior. Constituye la última capa de la red.

Los datos de entrada de la red neuronal suelen ser llamados *características* y es común agruparlas en un vector denotado por $x = [x_1, x_2, ..., x_n]$, donde n es el

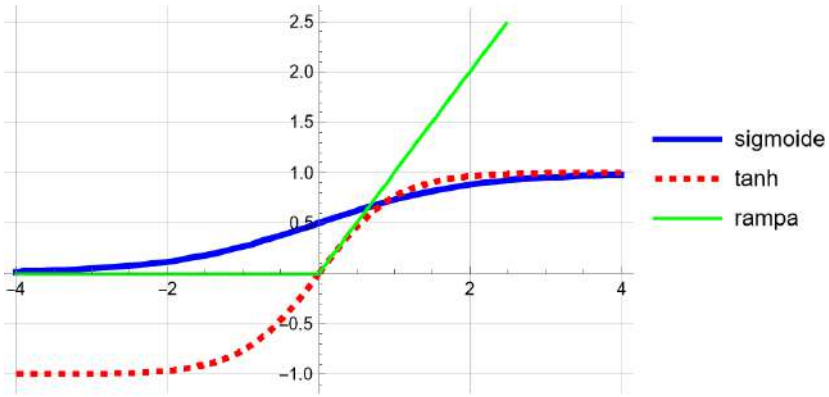


Figura 1.13: Las funciones de activación mas comúnmente utilizadas son las funciones sigmoidea, tangente hiperbólica y rampa, también conocida como ReLU (Rectified Linear Unit).

número de neuronas en la capa de entrada. Las salidas de las neuronas de las capas ocultas suelen agruparse en vectores denotados por $a_j = [a_{j,1}, a_{j,2}, ..., a_{j,n_j}]$, donde j es el nivel o número de capa y n_j es el número de neuronas en la j-ésima capa. Por su parte, la salida de la capa de salida suele denotarse por $\bar{y}$ y puede ser tanto un escalar como un vector.

Exceptuando las neuronas de la capa de entrada, la salida de cada neurona artificial es el resultado de evaluar una función matemática cuyo argumento es la combinación lineal de sus entradas, mismas que a su vez son ponderadas por una serie de parámetros internos denominados *pesos*. Los pesos de una capa j suelen ser agrupados en matrices denotadas por $\omega_j \in \mathbb{R}^{k \times n_j}$, donde k es el número de entradas de la capa y n_j es el número de neuronas o salidas en la capa. La función matemática que es evaluada es conocida como *función de activación* y es seleccionada cuidadosamente dependiendo de la aplicación de la red artificial (Figura 1.13). Las funciones de activación, como su propio nombre lo indica, determinan el nivel de activación que alcanza cada neurona artificial, en otras palabras, cuantifican la intensidad de la señal de salida de la neurona.

Dado un conjunto de datos de entrada o características x_T cuyo valor resultante y es conocido, la idea principal detrás de una red neuronal es ajustar sus pesos ω de tal manera que, al ingresar dicho conjunto de características a la red, el resultado o salida de la misma $\bar{y}$ se aproxime al valor previamente conocido, i.e., $\bar{y} - y \approx 0$. El valor y de un conjunto de características x suele recibir el nombre de *etiqueta*, a su vez, un conjunto de características también se conoce como ejemplo.

Idealmente, si los pesos de la red son adecuados, para un conjunto de características x_P cuyo valor resultante es desconocido, la red neuronal será capaz

de estimar un valor realista. Por ejemplo, suponiendo como caso de estudio la estimación del precio de casas, un posible conjunto de características estaría constituido por datos tales como el tamaño de la casa, su número de cuartos, número de baños, número de pisos, etc. Por otro lado, la etiqueta de este conjunto de características sería el precio o valor de la casa misma. Así, una red neuronal con pesos adecuados será capaz de estimar el precio de otras casas a partir del conjunto d características previamente mencionado.

El proceso de ajustar los pesos de una red neuronal se lleva a cabo mediante un proceso de "aprendizaje" denominado *entrenamiento*. La idea es proporcionar a la red neuronal un conjunto de ejemplos etiquetados de tal manera que cada valor obtenido por la red se aproxime a su etiqueta correspondiente.

El entrenamiento de una red neuronal sigue la siguiente secuencia:

- Un conjunto de ejemplos etiquetados, normalmente conocido como *conjunto de entrenamiento* $X_T = \left[x_T^1, x_T^2, \dots x_T^m\right]$, donde m es el número de ejemplos del conjunto de entrenamiento, es proporcionado a la red neuronal y se calcula el valor de salida correspondiente de cada ejemplo, i.e., $\bar{y}^i$, donde i hace referencia a i-ésimo ejemplo del conjunto de entrenamiento.

- Se calcula el error o diferencia entre cada resultado obtenido por la red $\bar{y}^i$ y su respectiva etiqueta y^i, i.e., $e^i = \bar{y}^i - y^i$. Posteriormente, mediante una función, denominada como *función de costo o pérdida*, se cuantifica la precisión de la red neuronal sobre todo el conjunto de entrenamiento.

  La función de costo, usualmente denotada como $J(\omega)$, no es única, pues la misma es selecciona dependiendo de la aplicación de la red neuronal. No obstante, una característica común y deseable de toda función de costo es que sea una función convexa (Figura 1.14). Esto se hace para asegurar que existe un mínimo global, es decir, que existe un único conjunto de pesos que aseguren que la función de costo tiene un mínimo valor posible.

  Una función de costo muy común es el Error Cuadrático Medio, definida como:

$$J(\omega) = \frac{1}{m} \sum_{i=1}^{m} \left(\bar{y}^i - y^i\right)^2 \tag{1.2}$$

- Mediante una técnica llamada *back propagation* [20] se calcula el gradiente de la función de costo respecto a cada peso de la red $\partial J(\omega)/\partial\omega$, es decir, se calcula como varía el valor de la función de costo conforme se modifican los pesos de la red.

- Mediante un algoritmo conocido como *Gradiente Descendiente*, se actualizan los pesos de la red. Esta actualización dependen del valor del gradiente de la función de costo, del valor de cada peso y de un parámetro conocido

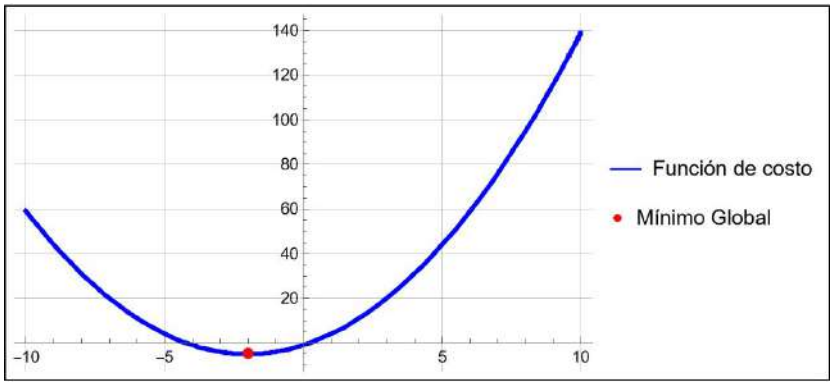


Figura 1.14: Una de las características mas notables de una función convexa es que presenta un mínimo global.

como *learning rate* o tasa de aprendizaje, denotado por α. En su forma más general, el algoritmo de gradiente descendiente es dado por:

$$\omega_{l+1} = \omega_l - \alpha \frac{\partial J(\omega)}{\partial \omega} \tag{1.3}$$

Donde l es el número de iteración o época, es decir, la cantidad de veces que el conjunto de entrenamiento ha sido mostrado a la red y por lo tanto que los parámetros de la misma han sido ajustados.

El objetivo del algoritmo de gradiente descendiente es minimizar la función de costo o el error entre la salida pronosticada y la etiqueta, hasta el punto que posteriores actualizaciones de los pesos no son necesarias.

La tasa de aprendizaje define el "tamaño" del paso que se da sobre la curva de la función de costo después de cada iteración para alcanzar el mínimo de la función misma. Su elección es un parámetro de diseño que suele elegirse en base a la pura experiencia del programador. Si bien con una tasa de aprendizaje grande se obtiene una convergencia acelerada, se corre el riesgo de sobrepasar el mínimo. Por el contrario, una tasa de aprendizaje pequeña tiene la ventaja de una mayor probabilidad de evitar sobrepasar el mínimo y por tanto de obtener una mayor precisión, no obstante, compromete la eficiencia general del proceso de entrenamiento, ya que requiere más tiempo y cálculos para alcanzar el mínimo de la función de costo. Por otro lado, el gradiente de la función de costo define la dirección en que se deben dar dichos pasos, es decir, si los pesos deben incrementarse o disminuirse.

- La secuencia anterior se repite hasta minimizar el valor de la función de costo, es decir, hasta obtener parámetros para los cuales la salida de cada ejemplo de entrenamiento se aproxime a su etiqueta correspondiente tanto como sea posible. Note que si el valor de la función de costo es igual a cero,

entonces para todo ejemplo del conjunto de entrenamiento se tendría que $y^i - \bar{y}^i = 0$.

Redes Neuronales Convolucionales

Las *redes neuronales convolucionales* (CNNs) son redes neuronales diseñadas para el tratamiento y segmentción de imágenes [21]. Se trata de redes neuronales artificiales que incorporan una serie de capas diseñadas para extraer características de una imagen de entrada. Estas capas son conocidas como *capas convolucionales* y su salida son una serie de características únicas que resultan mas adecuadas para una posterior segmentación o clasificación de la imagen.

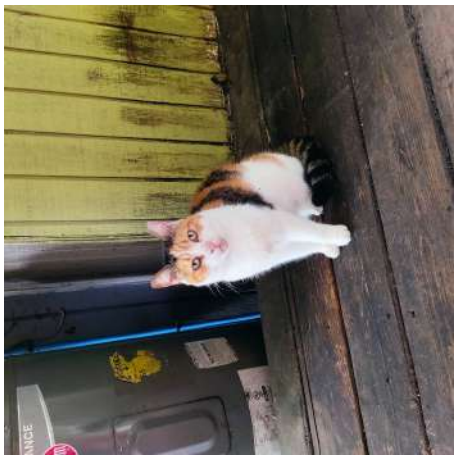
Cabe señalar que la forma de "alimetar" una red neuronal convolucional con una imagen es transformando el arreglo multidimensional de la imagen en una arreglo unidimensional, es decir, en un vector. En otras palabras, dada una imagen de dimensiones $m \times n$, el conjunto de características que ingresa a una red neuronal convolucional es un vector de longitud mn.

En el tratamiento de señales digitales, la convolución se utiliza para combinar dos señales con el propósito de conocer como va a cambiar la señal después de pasar por un determinado proceso. En el caso de analísis de imágenes, una convolución consiste en filtrar una imagen, esto es, se define un filtro o *kernel* (arreglo matricial) por el que se multiplica la matriz de la imagen. Dado que el kernel determina el resultado del filtrado, diferentes kernels generan diferentes resultados. No obstante, a grandes rasgos, el resultado del filtrado dependerá de las dimensiones kernel, así como de los valores que contiene.

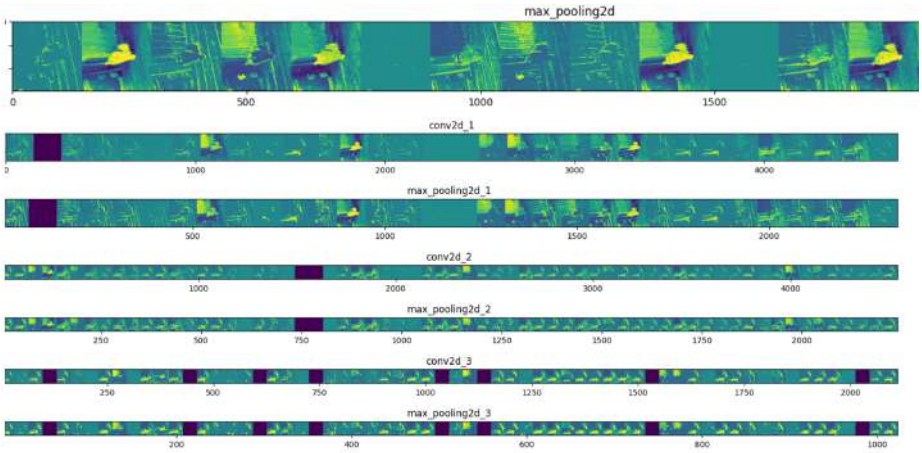
Tal como en el algoritmo de haarcascades, el propósito de las capas convolucionales es extraer las características mas relevantes de una imagen y luego usar dichas características para detectar o clasificar los objetos en una imagen. La gran diferencia es que las cacterísticas extraídas por las capas convolucionales son posteriormente procesadas por capas de neuronas convencionales, es decir, con sus respectivas funciones de activación y pesos.

Este posterior procesamiento de las características permite incrementar exponencialmente la diversidad de objetos que es posible identificar con una CNN. No obstante, esto también implica un incremento del costo computacional debido a la gran cantidad de características que hay que procesar.

Una forma de compensar el incremento del costo computacional es implementando capas de *pooling*. Las capas de pooling son elementos de las CNNs que se insertan generalmente a continuación de una capa convolucional y de la correspondiente función de activación. Su objetivo es realizar una selección de las características obtenidas por dicha capa, reduciendo las dimensiones de la imagen (Figura 1.15). Es decir, estas capas promedian vecindades de la imagen. En consecuencia, se preservan las características mas importantes y se reduce el



(a)



(b)

Figura 1.15: Procesamiento de una imagen en una CNN: a) imagen de entrada y b) resultado del procesamiento de la imagen después de atravesar una capa de convolución o de pooling.

tamaño de la imagen misma (dado que no es posible promediar las vecindades de los bordes y por tanto, estos se desechan). Es importante mencionar que aplicar demasiadas capas convolucionales o de pooling puede resultar en una pérdida de información innecesaria o más aún, en características insuficientes para el entrenamiento de la red.

1.5. Estimación de la profundidad

Como se ha mencionado anteriormente, la *estimación de la profundidad* es una tarea bastante popular y de suma importancia dada sus múltiples aplicaciones. La misma consiste en determinar la distancia a la que se encuentran los diversos objetos presentes en un espacio físico o escena 3D. Existen distintos métodos para la estimación de la profundidad, los cuales no necesariamente hacen uso de información visual. Dichos métodos de estimación, pueden clasificarse en dos grandes grupos:

1) *Métodos activos*: En estos métodos, el dispositivo de medición en cuestión envía algún tipo de señal a la superficie y posteriormente procesa la señal reflejada. Algunos ejemplos de métodos activos son los sonares y los sistemas LiDAR (Light Detection and Ranging), los cuales emiten sonido y haces de luz, respectivamente. Los métodos activos se caracterizan por ser bastante confiables, aunque suelen ser sumamente costosos y difíciles de montar y mantener.

2) *Métodos pasivos*: Por otro lado, los métodos pasivos basan sus mediciones en las señales emitidas de forma natural por la superficie en cuestión. Generalmente, estos métodos son de bajo costo y pueden ser montados con relativa facilidad en una gran variedad de escenarios. No obstante, suelen entregar mediciones menos precisas [22].

En la práctica, ambos métodos suelen ser combinados para obtener mejores mediciones. Ahora bien, claramente la visión artificial forman parte de los métodos pasivos. En particular, la utilización de arreglos de visión artificial binocular representan un método pasivo bastante utilizado para la estimación de la profundidad. Este método parte de la obtención de dos imágenes de una misma escena tomadas desde distintas posiciones (Figura 1.16). Posteriormente, dichas imágenes son procesadas y mediante *geometría proyectiva* se determina la profundidad.

Para determinar la profundidad con un arreglo de vision binocular es necesario cumplir con las siguientes subtareas:

1) Adquisición de imágenes.

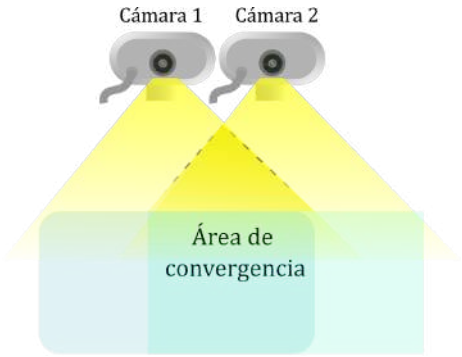


Figura 1.16: Sistema de visión artificial binocular para la estimación de profundidad. Al tener dos cámaras desplazadas, una con respecto a la otra, es posible obtener imágenes diferentes de la misma escena, mismas que pueden ser utilizadas para calcular la profundidad.

2) Calibración

3) Rectificación.

4) Correspondencia.

5) Extracción de la profundidad.

El objetivo de los primeros cuatro pasos es identificar las proyecciones en cada una de las imágenes de los puntos de interés en la escena tridimensional. Posteriormente, en el quinto paso se utiliza la información de dichas proyecciones para determinar la distancia a la que se halla el punto o puntos 3D del arreglo binocular.

1.5.1. Adquisición de imágenes

La captura o adquisición de imágenes por parte de arreglo binocular se debe realizar de forma simultánea, es decir, las imágenes capturadas, además de ser de la misma escena, deben pertenecer a un mismo instante.

Para conseguir lo anterior, es necesario *sincronizar las cámaras*, esto es, asegurar que ambas comiencen a capturar al mismo tiempo. Dado que idealmente las cámaras que conforman el arreglo binocular son idénticas, la *tasa de captura* o *frame rate* es o puede ser configurada para ser la misma en ambas, de tal manera que si las cámaras comienzan a capturar al mismo tiempo, los pares de imágenes posteriores corresponderán a un mismo instante.

La forma de sincronizar las cámaras se realiza mediante un *trigger* o *disparador*, es decir, una señal eléctrica enviada a ambas cámaras. La forma de enviar dicha señal, generalmente se realiza a través de un *puente eléctrico* entre las cámaras [23]. No obstante, también es posible implementar un trigger a través de un algoritmo. En dicho algoritmo, se inicializa una de las cámaras, dejándola en modo de espera hasta que la cámara restante inicie su proceso de captura [24]. El primer método es el mas eficiente, sin embargo, el segundo también es bastante utilizado por ser mas fácil de implementar.

Observación 1.1. *Estrictamente hablando, siempre existe un desfase de tiempo en la captura de imágenes, sin embargo, dicho desfase puede ser despreciado en la gran mayoría de aplicaciones. El problema de sincronización, es uno sumamente complicado y esta en constante estudio [25, 26].*

1.5.2. Calibración

Para poder deducir la posición tridimensional de un objeto, es necesario conocer antes algunas características muy importantes, estas son la geometría del arreglo binocular y los parámetros intrínsecos y extrínsecos de cada cámara [27]. Dependiendo de condiciones tales como la posición de la cámaras, su orientación en el espacio con respecto al sistema de coordenadas del mundo real, y la construcción misma de las cámaras, los parámetros intrínsecos y extrínsecos del arreglo binocular pueden cambiar. Para identificar estos parámetros e incluirlos en la posterior estimación de la profundidad, es necesario llevar a cabo un proceso de *calibración*.

El proceso de calibración consiste en:

1) Determinar los parámetros propios de las cámaras, tales como la longitud focal, la cuadratura de los píxeles, el centro de la imagen y la distorsión radial. La idea principal de identificar estos parámetros es corregir posibles distorsiones de la imagen [28].

2) Determinar la relación de posición relativa entre las dos cámaras. Esta relación está representada por una matriz de rotación y una matriz de traslación para cada cámara. Las cámaras, están referidas a un mismo sistema de coordenadas, por lo tanto, determinar la posición y orientación de una con respecto a la otra es relativamente sencillo.

Es importante aclarar que la distorsión de imagen es un fenómeno presente en toda cámara, no obstante, el grado de distorsión varia en función de la calidad de la cámara, y concretamente, depende de la calidad de su objetivo.

Existen dos tipos de distorsión: radial y tangencial [27]. La *distorsión radial* hace que los puntos en la imagen presenten cierto desplazamiento radial con

respecto al centro de la imagen. Este tipo de distorsión produce una pérdida de nitidez en los bordes de la imagen, así como una mayor curvatura de las lineas en la imagen, curvatura que incrementa conforme nos alejamos del centro de la imagen. Su principal causa es un centrado defectuoso del algún lente del objetivo. Es muy común en cámaras de gran angular y también en cámaras económicas debido a que es más barato producir lentes esféricas que parabólicas.

Por otro lado, la *distorsión tangencial* provoca que las rectas en la imagen que deberían ser paralelas entre sí, no lo sean. Su principal causa también es un centrado defectuoso de algún lente del objetivo. Ambos tipos de distorsiones pueden ser corregidas o al menos disminuidas después de un correcto proceso de calibración

Los parámetros identificados por el proceso de calibración son almacenados principalmente en tres matrices: la *matriz de rotación*, la *matriz de traslación* y la *matriz fundamental* (una para cada cámara). Las matrices de traslación y rotación simplemente describen las traslaciones y rotaciones en el espacio euclídeo tridimensional de una cámara con respecto a la otra. Por otro lado, la matriz fundamental almacena una serie de parámetros tales como la distancia focal y los coeficientes de distorsión. En otras palabras, las matrices de rotación y traslación contienen los parámetros extrínsecos del arreglo binocular mientras que la matriz fundamental contiene los intrínsecos.

1.5.3. Rectificación

Hasta ahora no se ha descrito a detalle el proceso para determinar la profundidad de un punto en el espacio a partir de un par de imágenes, no obstante, a grandes rasgos podemos decir que dicho proceso parte de la identificación de la proyección de dicho punto 3D sobre cada una de las imágenes. En otras palabras, consiste en identificar el par de píxeles que corresponden al mismo punto en el espacio. El proceso de identificar dichos píxeles se conoce como *correspondencia* (Figura 1.17).

Existen diferentes algoritmos de correspondencia, no obstante, dado que una imagen es un arreglo bidimensional, una practica inteligente consiste en limitar la búsqueda de píxeles correspondientes a una sola dimensión, es decir, es conveniente hallar imágenes alineadas horizontalmente tal que la búsqueda de píxeles correspondientes se restrinja a la misma linea horizontal. Alinear la imágenes horizontalmente se conoce como *rectificación* de imágenes.

La forma más sencilla de rectificar imágenes es rotando ambas cámaras hasta que estén mirando perpendicular a la línea que une a los centros de las cámaras, también conocida como *línea base*. Posteriormente, se hace que el eje vertical de cada cámara sea perpendicular a la línea base. Finalmente, puede ser necesario escalar las imágenes, esto es, tomar en cuenta la posibilidad de distancias focales

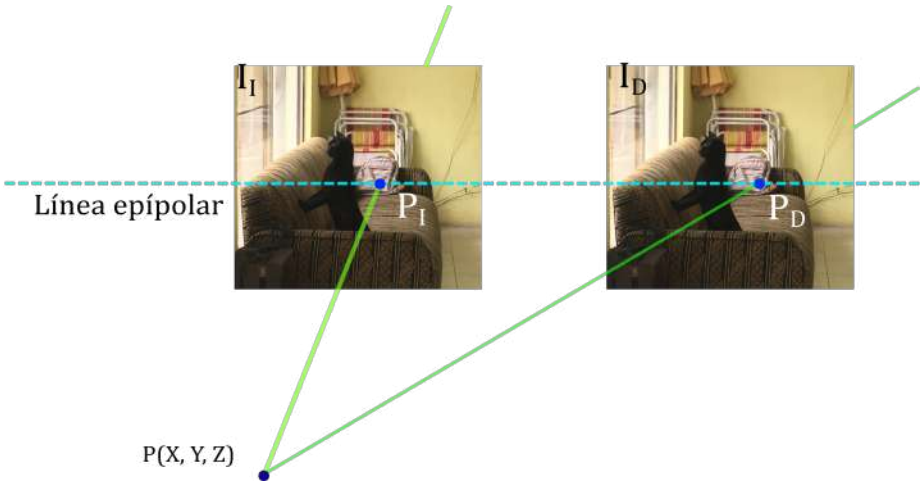


Figura 1.17: El proceso de búsqueda o identificación de píxeles que pertenecen a un mismo punto en el espacio, se conoce como *correspondencia*

diferentes [29].

1.5.4. Correspondencia

Los algoritmos de correspondencia se basan en dos tipos de técnicas:

- *Técnicas basadas en área:* Estas consideran un entorno o vecindad alrededor de un píxel en particular en una de las imágenes. Posteriormente, se analiza la imagen restante en busca de una región similar.

  Dado que el proceso anterior se repite para un número considerable de píxeles, es necesario hacer que el sistema de visión este sujeto a diferentes restricciones geométricas, tales como la que proporciona el proceso de rectificación.

- *Técnicas basadas en características:* En estas técnicas, la identificación de los píxeles se realiza tomando en cuenta determinadas características presentes en ambas imágenes. La idea es superponer ambas imágenes tomando en cuenta dichas características.

  Se entiende por *característica* a una estructura significativa en la imagen, tales como bordes, puntos de borde, segmentos de borde o regiones. Normalmente, la forma de identificar características consiste en analizar conjuntos de píxeles con atributos similares. Estos atributos pueden ser simples, como el color de lo píxeles o la intensidad de brillo.

Si bien las técnicas basadas en características pueden prescindir del proceso de rectificación, en la actualidad, las aplicaciones y los trabajos basados en técnicas de área predominan sobre las de caracterısticas. Por un lado, debido a que en muchas ocasiones las imágenes carecen de suficientes características bien definidas, resultando en correspondencias pobres. Por otro, las tareas de procesamiento que requieren las primeras son mas fáciles de implementar, lo cual se traduce en un menor costo computacional, posibilitando así su implementación en tiempo real.

Por otro lado, para determinar la similitud, y por lo tanto decidir si un par de píxeles, o en su defecto un par de características, corresponden a un mismo punto en el espacio, se emplean medidas estadísticas o métricas [30].

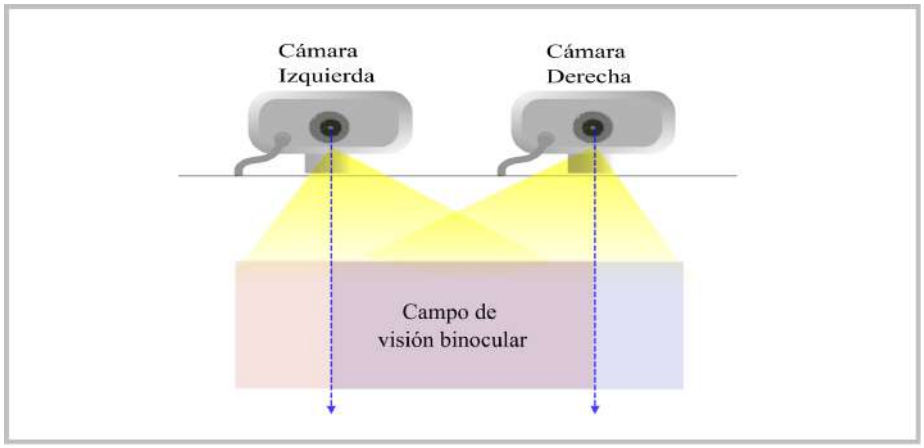
1.5.5. Extracción de la profundidad

La forma de obtener la distancia a la que se encuentra un objeto a partir de dos o mas imágenes digitales se basa en la geometría del arreglo de visión. Un sistema de visión binocular común se carácteriza por un par de cámaras (Logitech HD Pro C920) con sus ejes ópticos mutuamente paralelos y separados horizontalmente por una distancia que se denomina *línea base* y normalmente denotada por la letra b. El eje óptico de una cámara es la línea imaginaria que define el camino a lo largo del cual la luz se propaga a través de la misma, es decir, es una linea que une el centro de curvatura de cada lente del objetivo de la cámara y el centro de su sensor fotosensible, también conocido como *centro de proyección*. Las cámaras tienen sus ejes ópticos perpendiculares a la línea base y sus *líneas de exploración o epipolares* paralelas a la línea base. Una líneas epipolar es aquella que une las proyecciones en cada imagen de un mismo punto 3D (Figura 1.18) [27].

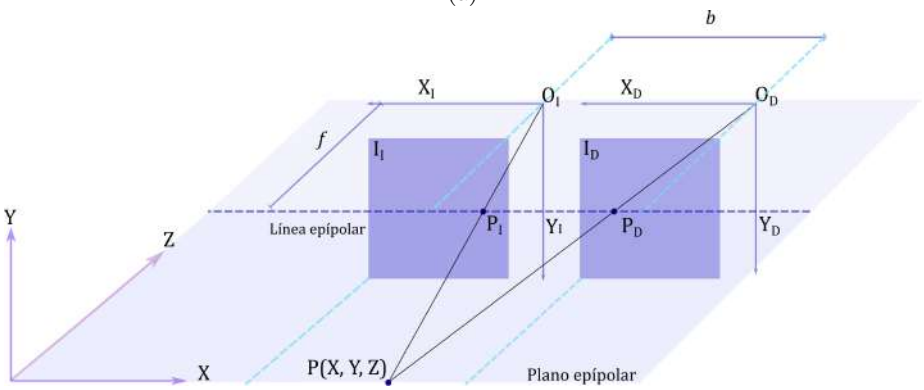
Como se mencionó anteriormente, comúnmente se constituye un arreglo binocular de tal manera que los ejes ópticos de las cámaras que lo conforman sean paralelos entre sí y perpendiculares a la linea base. Esta configuración se conoce como *restricción epipolar* y es sumamente conveniente pues simplifica en gran medida las relaciones para determinar la profundidad ya que limita el espacio de búsqueda de correspondencia. La restricción epipolar permite que la búsqueda de puntos correspondientes se limite a recorrer las imágenes por filas.

Ahora bien, una vez identificados los píxeles que corresponden a las proyecciones de un mismo punto en el espacio, la profundidad se puede determinar a partir de dichos puntos y analizando la *geometría epipolar* del arreglo. La geometría epipolar establece las relaciones geométricas que existen entre puntos espaciales en una escena y sus respectivas proyecciones en las imágenes [31].

Definición 1.1. ***Plano epipolar*** *[11]: Plano definido que contiene un punto de la escena 3D, sus respectivas proyecciones en ambas imágenes y los centros*



(a)



(b)

Figura 1.18: a) Configuración de un arreglo de visión artificial binocular y b) su respectiva geometría, también llamada geometría epipolar.

de proyección de ambas cámaras (centro del sensor fotosensible). Como consecuencia, los planos epipolares forman un haz de planos cuyo eje es precisamente la linea base. Cualquier punto del espacio tridimensional unido a los dos centros de proyección de las cámaras define este plano.

Definición 1.2. ***Epipolo*** *[11]: Punto en el que la linea base corta al plano de la imagen asociada. El epipolo es la proyección en una cámara del centro de proyección de la otra cámara. El epipolo puede caer fuera del trozo del plano de la imagen que devuelve la cámara. En particular, cuando la linea base es paralela al plano de la imagen, el epipolo es un punto en el infinito que puede ser representado por un punto del plano proyectivo.*

Definición 1.3. ***Línea epípolar*** *[11]: Es la intersección del plano epipolar con el plano de proyección de una cámara, esta línea une un mismo punto en la imagen izquierda y la imagen derecha. De manera que todos los puntos, cuyas proyecciones izquierdas se encuentren contenidas sobre una misma linea epipolar en la imagen derecha y viceversa.*

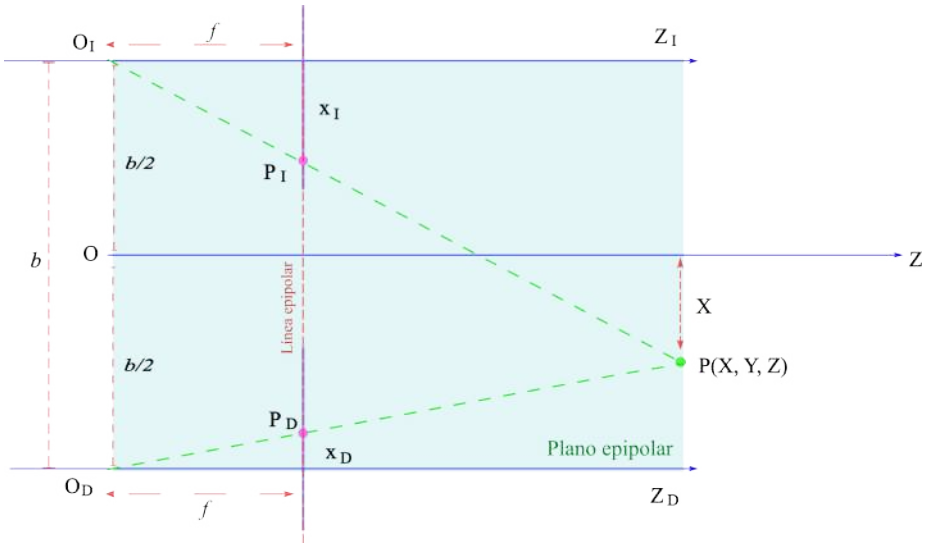


Figura 1.19: Geometría de dos cámaras con ejes ópticos paralelos desde una perspectiva superior.

Debido a la restricción epipolar, el calculo de la distancia se resume en una triangulación, es decir, en un análisis de triángulos semejantes considerando la geometría epipolar del arreglo binocular.

En el diagrama de la Figura 1.19, b es la línea base, es decir, es el segmento 3D que une los centros de proyección de ambas cámaras o en otras palabras es la distancia entre las cámaras. La distancia focal, denotada por f, es la distancia del lente al centro óptico/sensor óptico de la cámara (centro óptico izquierdo

O_I y derecho O_D). Ahora, dado un punto P en el espacio tridimensional con coordenadas (X, Y, Z), sus respectivas proyecciones en las imágenes derecha P_D e izquierda P_I tienen coordenadas x_D,y_D y x_I,y_I, respectivamente.

Del diagrama de la Figura 1.19, por triángulos semejantes tenemos que

$$\frac{\frac{b}{2} + X}{Z} = \frac{x_I}{f} \tag{1.4}$$

y

$$\frac{\frac{b}{2} - X}{Z} = \frac{x_D}{f} \tag{1.5}$$

Despejando ambas expresiones se tiene

$$x_I = \frac{f}{Z}\left(X + \frac{b}{2}\right) \tag{1.6}$$

$$x_D = \frac{f}{Z}\left(X - \frac{b}{2}\right) \tag{1.7}$$

Restando la expresión (1.7) de (1.6) se llega a

$$Z = \frac{f}{x_I - x_D} b \tag{1.8}$$

Definiendo la disparidad como $d = x_I - x_D$, entonces se tiene

$$Z = \frac{f}{d} b \tag{1.9}$$

La expresión (1.9) muestra que la profundidad Z es inversamente proporcional a la disparidad d, la cual es el desplazamiento horizontal del punto P proyectado.

1.6. Robot Operating System (ROS)

El *Sistema Operativo Robótico* o ROS (*Robot Operating System*) es un framework o marco de trabajo de desarrollo de software, es decir, proporciona un conjunto de librerías (herramientas de programación previamente desarrolladas) que permiten reciclar y desarrollar código para implementar nuevas aplicaciones en el ordenador. ROS provee los servicios estándar de un sistema operativo tales como abstracción del hardware, control de dispositivos de bajo nivel, implementación de funcionalidad de uso común, paso de mensajes entre procesos y mantenimiento de paquetes. Estas características permiten desarrollar, modificar y ejecutar códigos de programación a través y mediante uno o varios dispositivos con gran facilidad [32].

ROS se basa en una arquitectura distribuida de procesos, es decir, permite añadir nuevos servicios (funcionalidades) que pueden compartir los recursos disponibles con aquellos servicios ya existentes y sin perjudicarse entre sí. En ROS, estos servicios son generados en los llamados *nodos*. Un nodo es un archivo ejecutable que esta diseñado para controlar un elemento del hardware o desempeñar un proceso de cómputo. Todo nodo se ubica dentro de un *paquete* o *paquetería* ROS. Una paquetería es un conjunto de funcionalidades desarrolladas para desempeñar una tarea especifica. Dentro de ROS, los nodos pueden comunicarse con otros nodos a través de *mensajes* que transitan en canales de comunicación llamados *tópicos*. Las librerías y la arquitectura de ROS hacen posible trabajar con archivos ejecutables en distintos lenguajes de programación (e.g.: Python, C/C++, R, Java, etc.).

ROS es un software de libre acceso que puede ser utilizado tanto para aplicaciones comerciales como para fines de investigación. Mas importante aún es la importante comunidad alrededor de ROS, la cual continuamente contribuye con nuevas paqueterías y aplicaciones que pueden ser fácilmente integradas e incluso modificadas para desempeñar tareas personalizadas. Estas contribuciones suelen ser compartidas en plataformas en linea, principalmente en *repositorios* de GitHub [33], de donde pueden ser fácilmente descargadas.

Existen una serie de conceptos unicos en ROS, no obstante, los básicos son:

- **Master:** El ROS master o nodo maestro proporciona un registro que permite a el resto de nodos en la estructura encontrarse, intercambiar mensajes o solicitar servicios.
- **Nodo:** Un nodo es un archivo ejecutable. En ROS, generalmente coexisten varios nodos, los cuales comparten informacón entre si. Los nodos se comunican con el master, en un proceso de registro de informacón. A través de este proceso, los nodos pueden enviar y recibir información a y hacia los demás nodos.

- **Paquete o Paquetería:** Una paquetería ROS es una colección de nodos que esta diseñada para desempeñar una tarea especifica. En ROS, múltiples paqueterías pueden interactuar entre si.
- **Mensaje:** Es una estructura de datos escrita por algún nodo. Los mensajes se transmiten a través de un sistema de transporte constituido por múltiples canales de información.
- **Tema o tópico:** Se trata de un canal de comunicación donde transitan mensajes. Así, se dice que un nodo se *suscribe* a un tópico si lee los mensajes que transitan en este y *publica* si transmite mensajes a través de un tópico.
- **Espacio de trabajo o Workspace:** Se trata del directorio o folder principal donde se localizan las paqueterías ROS que se implementan. Por default, se designa como workspace al directorio de archivos de programa del ordenador (/usr en Linux y Program Files en Windows).
- **Catkin Workspace:** Se trata de un directorio externo al directorio de archivos de programa que puede ser utilizado como workspace. Aunque es posible construir paquetes ROS en el directorio de archivos de programa, lo recomendado es crear un Catkin workspace en el que es posible crear, modificar, instalar y compilar múltiples paqueterías de forma independiente. a la vez. Trabajar de esta manera permite tener la seguridad de que el directorio de archivos de programa de ROS no sera dañado en caso de que algo salga mal.
- **Launch:** En ROS cada nodo puede ser ejecutado por separado, no obstante, dado que diferentes nodos interactúan entre si, resulta mucho mas útil escribir un archivo que permita ejecutar varios nodos de forma simultánea. Este tipo de archivo se conoce como Launch.

En ROS es importante utilizar la línea de comandos (terminal en Linux y command prompt en Windows). Esta facilita al desarrollador la administración, visualización y recopilación de información. Si bien, se puede utilizar un entorno gráfico para interactuar con ROS, la gran mayoría de desarrolladores encuentran la linea de comandos muchos mas útil y rápida.

Existe una gran variedad de comandos para interactuar con ROS, no obstante, los mas comunmente utilizados son:

- roscore: Inicializa / ejecuta el nodo maestro.
- rosrun: Inicializa / ejecuta un nodo específico.
- roslaunch: Inicializa / ejecuta un launch especifico. Cabe señalar que al ejecutar un launch también se inicializa el nodo maestro.

- rosnode list: Enumera todos los nodos activos.
- rosnode info: Devuelve información sobre un nodo específico.
- rostopic list: Devuelve una lista de todos los tópicos activos.
- rostopic info: Permite obtener información sobre un tópico especifico.
- rostopic echo: Muestra la información contenida en un tópico, es decir, los mensajes.

Finalmente, cabe señalar que si bien ROS dispone de una versión para Windows, la gran mayoría de la comunidad alrededor de ROS trabaja en Linux. Por lo tanto, existe una mayor cantidad de paqueterías ROS para este sistema operativo, las cuales además suelen ser mas estables. Por lo tanto, es recomendable trabajar con Linux.

1.6.1. OpenCV

OpenCV (Biblioteca de visión artificial de código abierto) es una biblioteca de software de aprendizaje automático y visión artificial de código abierto para realizar aplicaciones de visión artificial en tiempo real. OpenCV se creó para proporcionar una infraestructura común para las aplicaciones de visión por computadora y para acelerar el uso de la percepción de la máquina en los productos comerciales. OpenCV esta orientado al procesamiento y analisis de imágenes. Hoy en día es una de las aplicaciones más usadas para el desarrollo de aplicaciones de visión artificial. [34]

La biblioteca de OpenCV tiene más de 2500 algoritmos optimizados, que incluyen un conjunto completo de algoritmos de aprendizaje automático y visión por computadora clásicos y de última generación. Estos algoritmos se pueden usar para:

- Reconocimiento facial
- Identificación de objetos o personas
- Inspección y vigilancia
- Recuento de objetos
- Róbotica
- Realidad aumentada

Existen paqueterías ROS optimizadas para incorporar todas las funcionalidades de OpenCV. Dado que OpenCV es de código abierto, dichas paqueterías pueden ser utilizadas gratuitamente tanto para fines comerciales como de investigación.

Capítulo 2

Sistema de visión binocular

Para describir el arreglo de visión artificial binocular implementado en este trabajo de tesis dividimos el mismo en:

- Hardware: el cual consta de dos cámaras con conectividad USB, un ordenador portátil, una tarjeta arduino, un controlador o driver, un motor a pasos y una base diseñada y parcialmente impresa en una impresora 3D para soportar las cámaras del arreglo.
- Software: constituido por todas las paqueterías de ROS, los códigos fuente de programación para el control del motor a pasos y para el procesamiento de imágenes, así como los archivos launch creados para facilitar la inicialización de todos los nodos ROS necesarios.

2.1. Descripción del hardware

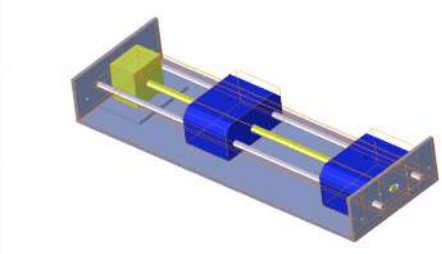


Figura 2.1: Dibujo asistido por computadora de la base propuesta para soportar las cámaras usb.

Para conformar el arreglo binocular, se propuso montar un par de cámaras USB sobre una base de fabricación propia y la cual permite modificar la separación entre las cámaras (linea base) con la ayuda de un motor a pasos. Distintos elementos de la base fueron llevados a impresión 3D para asegurar la mejor calidad posible de las piezas. De tal manera que se realizó un dibujo CAD (Figura 2.1).Por otro lado, la base contiene un par de varillas guía y un tornillo sin fin que permite el desplazamiento de una de las cámaras, es decir, permite transformar el movimiento angular del motor en movimiento lineal.

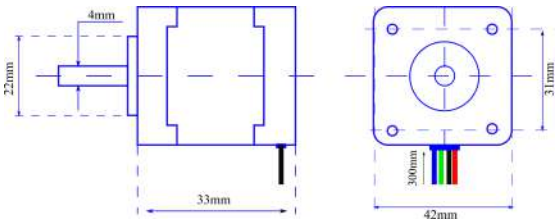


Figura 2.2: Un motor bipolar está representado por una salida de 4 hilos correspondiente a dos bobinas,

Los motores a paso se utilizan en una variedad de dispositivos que requieren un control de movimiento y posicionamiento preciso, que con el software adecuado, una computadora y un controlador, esto se puede realizar de manera simple. El motor a pasos utilizado fue uno de tipo bipolar **NEMA 17KS42STH34-15044** (las siglas del nombre hacen referencia al estándar industrial, utilizado para establecer normas mecánicas) que consta de dos fases, dos boninas expuestas por 4 cables y cuyas dimensiones son mostradas en la Figura 2.4a. Este motor necesita una alimentación de 12 V de corriente directa (DC) y su ángulo de paso indica cuantos grados se mueve el eje del rotor en cada paso. Las especificaciones completas del motor NEMA 17 se resumen en el Cuadro 2.1.

Ángulo de paso	1.8 (200pasos/revolución)
Voltaje	12VDC
Corriente	1500 mA
Resistencia	2.2 OHM
Torque	3.2 kg-cm
Dimensiones Figura 2.2	42 mm X 42 mm X 38 mm
Diámetro del eje	5 mm
Longitud de eje	22 mm
Longitud de cable	30 cm
No de fases	2

Cuadro 2.1: Especificaciones técnicas Motor del tipo bipolar NEMA 17.

El motor NEMA 17 es muy útil en diferentes aplicaciones como máquinas CNC o impresoras 3D. El ángulo de paso de este motor es de 1.8 grados, esto

es, el eje del rotor gira 1.8 grados en cada paso, lo que significa que puede dar 200 pasos por revolución.

Al tener como base un servomotor, el sistema mécanico obliga a que la electrónica genere señales PWM (Pulse Width Modulation) para su manejo, y teniendo en cuenta la comunicación con el computador, es necesario un enlace de tipo serial entre la electrónica y el dispositivo.

Dado que no es posible manejar el motor directamente desde una tarjeta Arduino, es necesario incorporar un controlador o driver. El driver elegido es el **DRIVER A4988**.

Figura 2.3: Controlador o driver modelo A4988

Se trata de un chip controlador A4988 que tiene varias funciones de seguridad incorporadas, como protección contra sobre-corriente, cortocircuito, bloqueo por baja tensión y sobre-temperatura. Cuya configutación de sus pines en alto o bajo controlan su funcionamiento; de manera que es posible configurar el ancho de paso del motor con los pines MSI, MS2 Y MS3, para activar o desactivar el controlador el botón ENABLE, La activacón del driver depende de la conexión de los botones RESET y SLEEP. Los pines STEP y DIR son configurados mediante el microcontrolador, GND es la conexión a tierra lógica. Las conexiones del con el motor son mediante los pines 1A, AB, 2A, 2B; VMOT es la conexión a la fuente de alimentación del motor, mientrsas que en VDD corresponde a la alimentación del driver. En la Figura 2.3 se pueden observar las posiciones respectivas de estos pines y las especificaciones completas de este controlador se encuentran en el Cuadro 2.2.

Tensión mínima de funcionamiento	8V
Tensión máxima de funcionamiento	35V
Corriente continua por fase	1A
Corriente máxima por fase	2A
Tensión lógica mínima	3V
Tensión lógica máxima	5.5V
Dimensiones	15.5 x 20.5mm

Cuadro 2.2: Especificaciones técnicas DRIVER A4988.

Existen diferentes formas de configurar un motor a pasos para conseguir un comportamiento deseado. Su configuración se realiza mediante Arduino, para lo cual se realizan los siguientes pasos:

1. Importar la biblioteca *AccelStepper*. AccelStepper mejora significativamente la biblioteca estándar de Arduino Stepper:
   - Soporta aceleración y desaceleración
   - Admite múltiples steppers simultáneos, con pasos simultáneos e independientes en cada stepper. La mayoría de las funciones de la API nunca se retrasan o bloquean (a menos que se indique lo contrario)
   - Admite pasos de 2, 3 y 4 cables, además de medios steppers de 3 y 4 cables.
   - Permite admitir velocidades muy lentas.
   - Proporciona una API extensa con funciones más sofisticadas.
   - Da mayor compatibilidad con subclases.
2. Definir los pines de entrada y salida que se utilizaran para controlar el motor. En el programa del microcontrolador se define el movimiento del motor deseado, esto es, hacia la izquierda y la derecha, respectivamente, así como la velocidad de giro.
3. Crear una instancia de la clase AccelStepper y configurar los parámetros del motor, como el número de pasos por revolución y la velocidad máxima.
4. Utilizar los métodos de la clase AccelStepper para controlar el movimiento del motor, como moveTo() para mover el motor a una posición especifica y run() para hacer que el motor gire.

El código utilizado para controlar el motor a pasos puede ser encontrado en el apéndice. Por otro lado, el circuito que permite configurar el motor en conjunto con la tarjeta Arduino y el driver se muestra en la Figura 2.4.

Respecto a las cámaras USB para el arreglo binocular, se contemplaron diferentes opciones, principalmente las mostradas en la Tabla 2.3. Las cámaras en dicha tabla presentan una calidad de imagen de hasta 1080 píxeles, tasa de captura de 5 hasta 30 fps y un costo accesible. El factor determinante al momento de elegir alguna de estas opciones fue la posibilidad de configurar la taza de captura y calidad de imagen. Esta características era deseable dado que se requirió configurar las cámaras USB tal que el ordenador disponible fuera capaz de procesar la información capturada con fluidez.

Las características del ordenador utilizado son mostradas en el Cuadro 2.4. Se trata de una computadora portátil con un procesador i7 de sexta generación

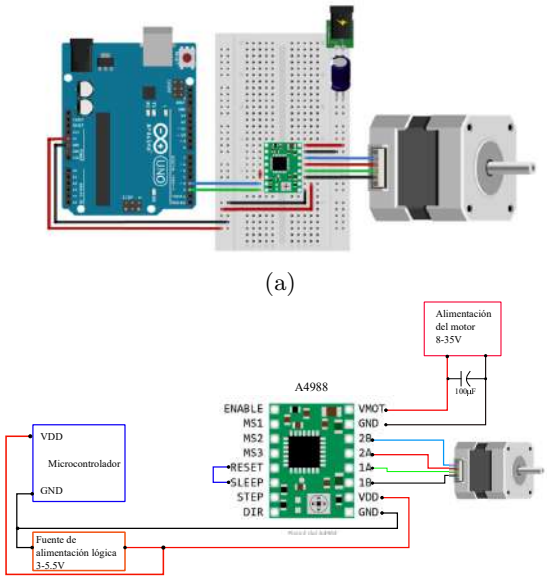


Figura 2.4: Diagrama del circuito encargado de controlar el motor a pasos.

con 2 núcleos. Cabe senãlar que el ordenador posee una GPU integrada, por lo que su capacidad de procesamiento es limitada. Así, finalmente se optó por las cámaras USB Logitech HD Pro C920, capaces de capturar imágenes de hasta 1080 píxeles y configurables para capturar imágenes de menor calidad y con una tasa de captura variable de 5 hasta 30 fps. Una desventaja de estas cámaras es que no poseen tecnología USB2.0, por lo que la velocidad de trasmisión de información es inferior al resto de opciones.

Cámaras		
Modelo	PC ZZCP	Logitech HD Pro C920
Resolución de captura	1080p	1080p - 720p
Tasa de captura	30 fps	5-30 fps
Tegnología	USB 2.0	USB
Longitud focal mínima	1	

Cuadro 2.3: Especificaciones técnicas de cámaras USB candidatas para el arreglo binocular.

Laptop	
Modelo	Inspiron 157569
Procesador	Intel ®Core ™i7-6500UCPU 2.50GHzx4
Gráficos	Mesa Intel®HD Graphics 520 (SKL GT2)
Memoria	11.6 GB
Capacidad del disco	512.1 GB

Cuadro 2.4: Especificaciones técnicas de ordenador portátil utilizado.

2.2. Descripción del Software

ROS consiste en una basta colección de bibliotecas desarrolladas para una gran variedad de aplicaciones en robótica. Muchas de estas bibliotecas son creadas y mantenidas por el Instituto de Willow Garage, creadores originales de ROS en la Universidad de Stanford. No obstante, una gran cantidad de bibliotecas son contribuciones de la basta comunidad alrededor de ROS. Estas bibliotecas suelen ser de código abierto, es decir, cualquier usuario puede aplicarlas directamente o utilizarlas como base para nuevos proyectos. En el capítulo anterior se mencionó que un espacio de trabajo Catkin es una carpeta o directorio donde se instalan, crean y modifican paqueterías de ROS sin la necesidad de alterar las bibliotecas base del mismo, ubicadas en el directorio de instalación de ROS.

Crear un espacio de trabajo Catkin es posible dado que ROS utiliza un principio de superposición donde el esquema de compilación puede encontrar dependencias en diferentes paqueterías instaladas en distintos directorios del sistema. En particular, un espacio de trabajo Catkin incluye un conjunto de configuraciones de entorno que le permiten hallar y utilizar cualquier recurso de ROS que se halle instalado en el sistema.

Algo importante a señalar sobre ROS es que existen versiones del mismo capaces de ejecutarse en Windows, no obstante, dado que la gran mayoría de la comunidad alrededor de ROS trabaja en sistemas operativos Linux, tales como Ubuntu, Fedora, RedHat, etc., es conveniente hacer uso de alguna de estas distribuciones. De tal manera que en este trabajo de tesis se hará uso de una distribución ROS para Linux y se creará un espacio de trabajo Catkin, evitando así posibles daños al directorio de instalación de ROS.

A continuación resumimos los pasos seguidos para configurar ROS y el espacio de trabajo Catkin en un sistema operativo Linux. Más detalles sobre esto pueden hallarse en la página oficial de ROS: *https://www.ros.org/*

1. **Instalación de ROS**. Para este paso, es preciso seguir las instrucciones tal y como se describen en: http://wiki.ros.org/ROS/Installation

Hay varias distribuciones de ROS para Linux. No obstante, la versión que se utilizó para el desarrollo de este trabajo es una anterior a la mas reciente. Se decidió esto dado que las últimas versiones de ROS suelen ser inestables. La distribución elegida fue **ROS Noetic Ninjemys**, lanzada en Mayo de 2020 y con soporte hasta Mayo 2025. Adicionalemente, esta distribución es recomendada para Ubuntu 20.04.

La instalación de ROS consiste en una serie de comandos que deben ser introducidos en la terminal de nuestro sistema operativo Linux, en este caso Ubuntu 20.04. El comando fundamental para la instalción es:

```
sudo apt install ros-noetic-desktop-full
```

Este comando inicia la descarga e instalación de ROS. Finalizada la instalación, es posible ejecutar ROS con el comando

```
roscore
```

2. **Creación de espacio de trabajo Catkin**. Para crear un espacio de trabajo, es necesario primero saber que este consta de:

   - Un archivo package.xml para describir el contenido del espacio de trabajo.
   - Un archivo CMakelist.txt que define las reglas de compilación del espacio de trabajo.
   - Directorio individual para cada paquetería instalada en el espacio de trabajo.

   Las paqueterías en un espacio Catkin se pueden compilar como proyectos independientes o como un todo, es decir, los espacios Catkin proporcionan herramientas para compilar múltiples paqueterías a la vez o de forma independiente.

   Para crear un espacio de trabajo Catkin, se hace uso de los siguientes comandos en la terminal:

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/
catkin_make
```

   El primer comando crea un directorio vacío llamado catkin_ws (este puede llamarse como nosotros deseemos). Posteriormente, con cd nos desplazamos al interior del directorio recientemente creado para finalmente compilar nuestro espacio de trabajo con catkin_make. Más detalles sobre como configurar por completo un espacio de trabajo Catkin pueden hallarse en $http://wiki.ros.org/catkin/Tutorials/create\_a\_workspace$.

3. **Instalación de paqueterías ROS**. Creado y compilado nuestro espacio de trabajo, podemos ahora instalar en este todas aquellas paqueterías necesarias. El proceso para instalar un paquetería en el espacio de trabajo Catkin consiste en 1) descargar (clonar) la paquetería localizada en

algún repositorio en línea. Generalmente, dichos repositorios se localizan en GitHub: *https* : *//github.com/*. Si la paquetería en cuestión se halla en un repositorio de GitHub utilizamos el siguiente comando para clonar la paquetería:

```
git clone -b <nombre paqueteria> <url de repositorio>
```

Una vez descargada la paquetería, antes de poder hacer uso de esta es necesario compilar nuevamente el espacio de trabajo con:

```
cd ~/catkin_ws/src
catkin_make
```

A continuación se enlistan las paqueterías que fueron instaladas en el espacio Catkin del presente trabajo de tesis.

- **usb_cam:** Paquetería que proporciona una interfaz configurable que permite la comunicación entre ROS y el controlador de las cámaras web USB estándar, es decir, permite transmitir las imágenes capturadas por las cámaras USB a ROS. Con esta paquetería también es posible consultar las características principales de una cámara USB así como realizar algunas configuraciones básicas (si la cámara USB lo permite), tales como la tasa de captura (frame rate), el espacio de color o el tamaño de la imagen.

  La configuración de una cámara USB puede especificarse directamente en el código fuente del nodo ROS, en un archivo launch o con la ayuda de la paquetería *dynamic_reconfigure*:
  http : *//wiki.ros.org/dynamic_reconfigure*.

  Una descripción mucho mas detallada, así como los enlaces al repositorio donde se halla localizada esta paquetería se pueden consultar en: *http* : *//wiki.ros.org/usb_cam*.

- **image_pipeline:** Se trata de un conjunto de paqueterías ROS específicamente diseñadas para el procesamiento de imágenes. Las paqueterías que conforman image_pipeline son:

  - camera_calibration
  - depth_image_proc
  - image_proc
  - image_publisher
  - image_rotate
  - image_view
  - stereo_image_proc.

  Una descripción detallada de image_pipeline así como los enlaces a paqueterias que la conforman se pueden hallar en:
  http : *//wiki.ros.org/image_pipeline*.

Para este trabajo de tesis, se hace uso de stereo_image_proc para un procesamiento básico de las imagenes de un par binocular, camera_calibration para la calibración del par binocular e image_view que permite la visualización de cualquier tópico con imagenes.

- **stereo_image_proc:** Esta paquetería contiene los nodos necesarios para comunicar los nodos de control de las cámaras USB, los nodos de procesamiento de visión y los datos de calibración de las cámaras. Esta paquetería hace uso de datos de calibración para eliminar la distorsión de las imágenes, así como para rectificar las mismas, es decir, arroja imágenes cuyas líneas de exploración se alinean, tal que un posterior procesamiento es mas efectivo y rápido.

  Adicionalmente, este nodo es capaz de producir una nube de puntos que se puede visualizar con ayuda de una paquetería adicional llamada *rviz*. No obstante, en este trabajo de tesis no se hace uso de rviz ni de dicha nube de puntos. La documentación y el código fuente de la paquetería stereo_image_proc pueden hallarse en: $http://wiki.ros.org/stereo\_image\_proc$.
- **camera_calibration:** La paquetería de camera_calibration permite una fácil calibración de cámaras monoculares (ya sea de forma individual o en conjunto) utilizando un objetivo de tablero, tal y como el que se muestra en la Figura 2.5. De forma predeterminada, la paquetería asume una sincronía de las cámaras, esto es, que las mismas se activan simultáneamente. El proceso de calibración es ilustrado posteriormente en este capítulo, no obstante, dicho proceso, la documentación y el repositorio de camera_calibration pueden hallarse en: $http://wiki.ros.org/camera\_calibration$
- **image_view:** Esta paquetería permite visualizar las imágenes generadas por cada nodo que envié mensajes de este tipo, es decir, permite desde visualizar las imagenes originales captadas por las cámaras USB hasta las imágenes rectificadas enviadas por stereo_image_proc. Es una paquetería bastante simple que no interviene en cuestiones de procesamiento, simplemente se limita a permitir una visualización de los resultados. Su documentación y código fuente se halla en: $https://wiki.ros.org/image\_view$.

Las paqueterías mencionadas anteriormente son comúnes para todas las técnicas de detección de objetos implementadas en este trabajo de tesis. Las paquetería específicas de cada técnica serán mencionadas posteriormente.

2.2.1. Calibración del arreglo binocular

Como se mencionó anteriormente, independientemente de la técnica de detección de objetos implementada, antes es necesario calibrar el arreglo binocular,

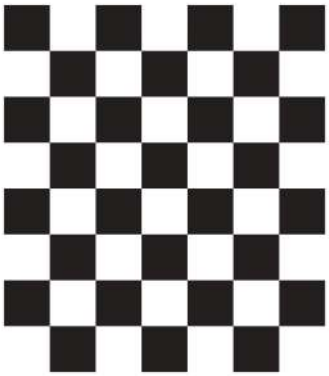


Figura 2.5: Para la calibración se utiliza una cuadrícula cuyas esquinas se utilizan como puntos de referencia junto con las dimensiones de la cuadrícula misma.

pues esta influye en gran medida en los resultados. A continuación se describe como se efectúa este procedimiento con ayuda de las paqueterías correspondientes de ROS.

El proceso de calibración de un arreglo binocular consiste básicamente en compensar los defectos generados por las características de las cámaras del par binocular (hallar parámetros intrínsecos) y los defectos producidos por la geometría del arreglo en sí (parámetros extrínsecos). Esto se consigue con la ayuda de un patrón geométrico y la identificación de puntos de referencia en este. De hecho, las técnicas de calibración suelen clasificarse según el tipo de patrón utilizado. El patrón mas conocido es el llamado *patrón planar*, mismo que asemeja a un tablero de ajedrez (Figura 2.5). En este tipo de patrón o tablero, las esquinas de la cuadrícula junto con el tamanõ del tablero son utilizadas como referencia.

Procedimiento de calibración con ROS y OpenCV

Para calibrar las cámaras, nos apoyamos en la paquetería camera_calibration, la cual esta contenida en el directorio de instalación de ROS, pero es aconsejable instalarla en el espacio de trabajo Catkin en caso de requerir realizar alguna modificación.

1) **Inicialización de espacio de trabajo y cámaras USB:** Para iniciar con la calibración del par binocular se debe inicializar primero el espacio de trabajo Catkin en una terminal de Linux, posteriormente, en la misma terminal podemos inicializar los nodos responsables de controlar a las cámaras USB. Después de ejecutar dichos nodos, se crean los tópicos correspondientes con la información de las imágenes originales (image_raw), es decir, imágenes sin procesar (Figura 2.6). Para inicializar el espacio de

trabajo y los nodos de las cámaras USB se ejecuta la siguiente serie de comandos:

```
cd < nombre del espacio de trabajo >
source devel\setup.bash
roslaunch usb_cam usb_cam_2cam.launch
```

Note que los nodos de las cámaras se inicializan simultánemente con ayuda de un archivo launch (ver Apéndice, Cuadro 3.15).

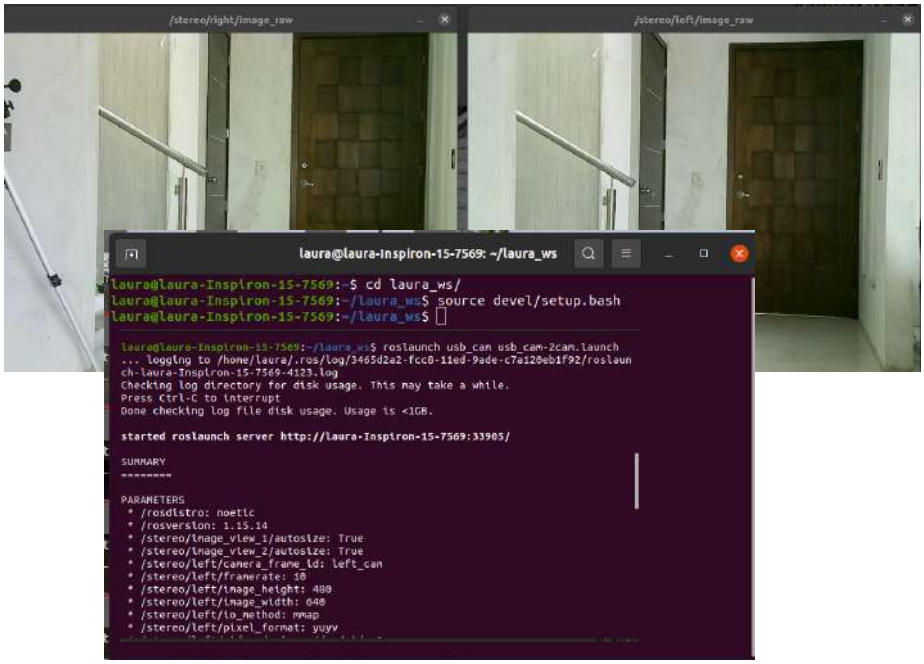


Figura 2.6: En la terminal de Linux, inicializamos los nodos usb_cam, posteriormente, en otra terminal ejecutamos el nodo de calibración cameracalibrator.py.

2) **Inicialización de nodo de calibración:** Para la calibración del sistema binocular, la paquetería camera_calibration ofrece el nodo cameracalibrator.py, el cual aprovecha muchas de las funcionalidades de OpenCV. El nodo cameracalibrator.py se suscribe a los tópicos de las imágenes sin procesar (image_raw) y despliega una ventana guía para la calibración. Para dar inicio a la calibración, ejecutamos el siguiente comando en una terminal nueva:

```
rosrun camera_calibration cameracalibrator.py --approximate
    0.1 --size 7x6 --square 0.71 right:=/stereo/right/
    image_raw left:=/stereo/left/image_raw right_camera:=/
    stereo/right left_camera:=/stereo/left
```

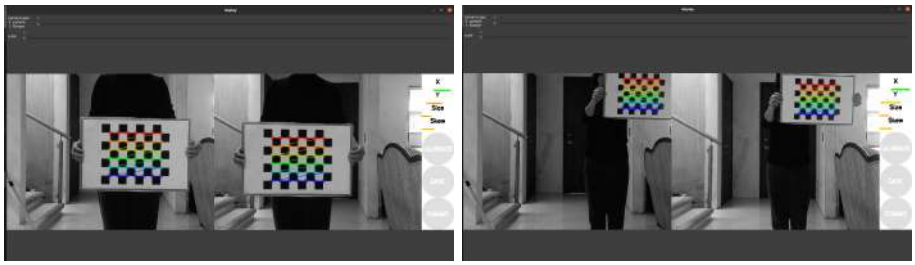
El comando anterior inicializa el nodo cameracalibratorpy bajo una determinada configuración, en este caso, se especifica el nombre de los tópicos de

Figura 2.7: Una vez inicializados los nodos responsables de controlar las cámaras USB, en una nueva terminal inicializamos el nodo cameracalibrator.py para comenzar la calibración.

las imagenes así como las dimensiones y el número de cuadros del tablero que se utilizará para la calibración (Figura 2.7). El nodo de calibración incluye un conjunto de algoritmos de procesamiento de imágenes, tales como: detección de bordes, detección de esquinas, manchas, etc. Estas funciones permiten obtener los parámetros necesarios para la correcta calibración del par binocular. Dichos algoritmos hacen uso del patrón geométrico y necesitan al menos diez imágenes del mismo en diferentes posiciones. A estas imágenes se les hace una transformación previa a escala de grises para que puedan ser tratadas con mayor facilidad por las funciones encontradas en las librerías de Opencv.

3) **Rotación y traslación de tablero:** El nodo de calibración despliega una ventana que muestra el campo de visión del par binocular. Se procede con la calibración sosteniendo el tablero horizontalmente de tal manera que es visible para ambas cámaras. Para obtener una buena calibración se debe mover a lo largo y ancho del campo de visión (Figura 2.8), esto es:

   - Calibración en X: el tablero se detecta en los bordes izquierdo y derecho.
   - Calibración en Y : el tablero se detecta en los borde superior e inferior.
   - Skew: el tablero se detecta con varios ángulos de rotación con respecto a la cámaras.
   - Calibración de tamaño: el tablero se acerca o aleja tal que ocupe una mayor o menor cantidad del campo de visión.



(a) (b)

(c)

Figura 2.8: Para una correcta calibración el tablero debe a) ocupar gran parte del campo de visión, b) ser detectado en todos los bordes: izquierdo, derecho, superior e inferior, c) ser detectado con diferentes ángulos de rotación con respecto al par binocular.

La ventana desplegada por el nodo de calibración muestra una serie de barras indicadoras que aumentaran de longitud a medida que el tablero se mueve. Una vez que se han obtenido suficientes datos de calibración, se activa el botón *CALIBRATE*, lo que indica que el nodo ha recolectado suficiente información para la calibración del par binocular. Por otro lado, la opción *COMMINT* guarda de forma permanente los datos de calibración en el buffer de cada cámara. A su vez, el botón *SAVE* genera una carpeta con los datos de calibración.

4) **Resultados de calibración:** Una vez finalizado el proceso de captura y procesamiento de diferentes posiciones del tablero, se obtiene una serie de arreglos numéricos con los parámetros intrínsecos y extrínsecos de ambas cámaras. Estos datos son salvados en un par de archivos con extensión yaml al hacer click en el botón *SAVE*. El contenido de cada archivo .yaml consiste en:

- Matriz de la cámara:

$$\mathbf{K} = \begin{bmatrix} f_x & \Upsilon & c_x \\ a & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

Donde los valores de f_x, f_y, son los valores de la distancia focal en x, y. Los valores que corresponden a las coordenadas del centro óptico de la imagen corresponden a c_x, c_y. El valor de Υ que por lo regular es 0, equivale al sesgo entre los ejes. En el archivo .yaml esta matriz se halla al inicio del mismo y luce de la siguiente manera:

```
image_width: 640
image_height: 480
camera_name: narrow_stereo/left
camera_matrix:
  rows: 3
  cols: 3
  data: [839.64296,   0.     , 346.08006,
           0.     , 855.58399, 125.21412,
           0.     ,   0.     ,   1.     ]
```

Note como el ancho y altura de la imagen, así como el nombre de la cámara, preceden a la matriz de la cámara en el archivo .yaml.

- Coeficientes de distorsión:

$$\mathbf{D} = \begin{bmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{bmatrix} \tag{2.2}$$

Los valores k_1, k_2 y k_3 son coeficientes de distorsión radial, mientras que p_1 y p_2 son coeficientes de distorsión tangencial. En el archivo .yaml, el vector que contiene estos coeficientes luce como se muestra a continuación:

```
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [0.038488, 0.020682, -0.039695, -0.001658,
    0.000000]
```

Observe que el modelo de distorsión utilizado para calcular los coeficientes es especificado en el archivo .yaml. En este caso se trata del modelo de distorsión de Brown [35].

- Matriz de rectificación:

  La matriz de rectificación es una matriz que describe la rotación necesaria para que las lineas epipolares del par binocular se alineen tal que estas sean paralelas en el plano de visión binocular. En el archivo .yaml esta matriz luce como:

```
rectification_matrix:
  rows: 3
  cols: 3
  data: [ 0.99706048,  0.00541418,  0.07642701,
         -0.0044608 ,  0.99991017, -0.01263959,
         -0.07648858,  0.01226151,  0.99699506]
```

Se puede notar que los elementos de la diagonal principal son prácticamente uno, es decir, la rotación necesaria para alinear las lineas epipolares es mínima.

- Matriz de proyección:
  Los valores de la matriz de preyección definen una matriz de rotación $\mathbf{R}$ de 3×3 y un vector de traslación $\mathbf{t}$ de tres elementos. La matriz de proyección se representa de la siguiente forma:

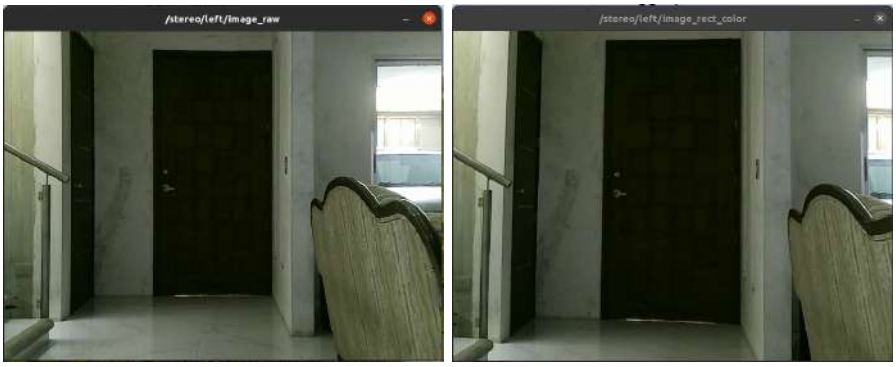
$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \mathbf{R} \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} + \mathbf{t} \tag{2.3}$$

  Para conformar la matriz de proyección, se agrega el vector de traslación como una columna al final de la matriz de rotación, de tal manera que se obtiene una matriz de 3×4.

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = [\mathbf{R}|\mathbf{t}] \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} \tag{2.4}$$

  La matriz de proyección en el archivo .yaml luce de la siguiente manera:

```
projection_matrix:
  rows: 3
  cols: 4
  data: [909.0119 ,   0.     , 265.98412,   0.     ,
           0.     , 909.0119 , 130.43674,   0.     ,
           0.     ,   0.     ,   1.     ,   0.     ]
```



(a) (b)

Figura 2.9: Calibración de imágenes: a) Imagen original y b) imagen calibrada. Observe como la distorsión radial y tangencial es menor en b), esto es, la curvatura de las líneas, que en realidad son rectas, disminuye.

Para llevar a cabo la calibración del par binocular, dado que se requiere ejecutar varios nodos, lo más conveniente en ROS es crear un documento o archivo launch. Como se mencionó anteriormente, este tipo de archivo

ejecutable permite inicializar de forma secuencial varios nodos a través de un simple comando en la terminal. El contenido del archivo launch correspondiente puede hallarse en el apéndice de este documento. Para ejecurar un archivo launch desde la terminal, se hace uso del siguiente comando:

```
$ roslaunch <nombre de la paqueteria> <nombre del archivo
>.launch
```

El comando roslaunch examina automáticamente el paquete y detecta los archivos de inicio disponibles. La rectificación de las imágenes en las cámaras la realiza el nodo, stereo_image_proc, de la paquetería stereo_image_proc; este nodo publica imágenes rectificadas en el tópico image_rect_color (Figura 2.9).

Como se mencionó, después de la calibración, los resultados de la misma son almacenados en el buffer de las cámaras, no obstante también es posible guardar dichos datos en un archivo .yalm. Esto permite hacer uso de distintas calibraciones simplemente especificando la ruta a dichos archivos. Esta instrucción puede incluir en un archivo launch para su posterior consideración:

```
<param name="camera_info_url" value="file:///home/laura/
Escritorio/CALIBRACIONMAYO/Lb18cm_2105/left.yaml"/>
```

2.3. Algorítmos de detección de objetos

Al visualizar imágenes, vídeos, para el ser humano resulta sencillo localizar e identificar fácilmente un objeto de interés. La identificación de objetos no es más que la transmisión de esta habilidad a una computadora. La detección de objetos ha encontrado su aplicación en una variedad de campos, como la videovigilancia y los vehículos de navegación autónoma. Aunque en la actualidad se utilizan cada vez más las redes neuronales para resolver problemas de visión por computadora. En este trabajo se realizó una revisi'on de los algoritmos de detección de objetos que se presenta para comprobar si los distintos métodos utilizados proporcionaban resultados similares.

2.3.1. Identificación de objetos por color

La técnica de detección de objetos mediante segmentación de color esta implementada en la paquetería stereo_color_tracker. Esta paqueteria contiene el nodo tracker_node encargado de implementar las bibliotecas de OpenCV necesarias para segmentar la imagen de acuerdo a un rango de colores predefinido por el usuario. La línea de código en el archivo launch responsable de inicializar el nodo tracker_node es la siguiente:

```
<node name="tracker_node" pkg="stereo_color_tracker" type="
    track_color.py" output="screen">
```

Rango de colores para la segmentación: Para determinar el rango de colores que el algoritmo de segmentación toma en consideración, la paquetería stereo_color_tracker incluye un script de python llamado range_detector.py. Al ejecutar dicho script, se abre una ventana como la mostrada en la Figura 2.10.

(a) (b)

Figura 2.10: Segmentación de color: determinación del rango de colores a considerar para la extracción de características de intéres en una imagen.

Hacer uso de este script y determinar el rango de colores es bastante sencillo y consiste en los siguientes pasos:

1. Tomar una foto, del objeto. Esta foto debe ser clara, de preferencia centrado el objeto y en el fondo de la escena no debe haber objetos del mismo color.
2. Para poder trabajar con esta imagen, el archivo debe ubicarse en la carpeta del espacio de trabajo.
3. Iniciar el nodo maestro de ROS en una terminal.
4. En otra terminal, después de haber inicializado el espacio de trabajo; ejecutar la siguiente orden

   ```
   python3 range_detector.py --image [nombredelaimagen].jpeg --
       filter HSV --preview.
   ```

   A notar que la instrucción anterior incluye el nombre de la imagen anteriormente capturada.
5. Después de ejecutar esta orden en la terminal, el programa abrirá dos ventanas, una de ellas nos permite ajustar los valores de rango de color en el espacio de color HSV. Es posible manipular estos valores hasta aislar el color que se quiera.
6. Despues de hallar los valores que aislan el color, los mismos son incluidos en el código del nodo tracker_node, esto es, en el archivo track_color.py.

Una vez que el rango de colores es proporcionado al algoritmo en el espacio de color HSV, se aislara todo objeto en la imagen que pertenezca a dicho rango.

2.3.2. Identificación de objetos con Haarcascades

OpenCV proporciona modelos clasificadores en cascada Haar, previamente entrenados con un conjunto de datos positivos y negativos. Estos se pueden encontrar en la carpeta de instalación de OpenCV (Figura 2.11) o en el siguiente repositorio de GitHub:

https : //github.com/opencv/opencv/tree/master/data/haarcascades

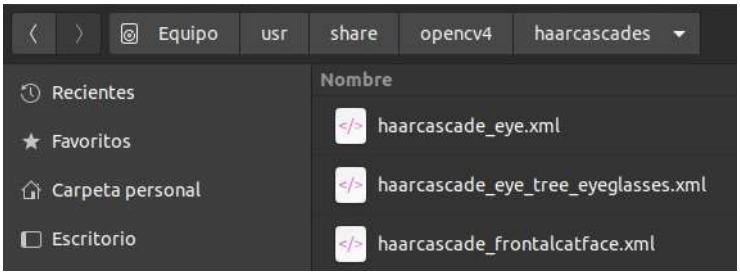


Figura 2.11: Ruta de archivos xml (Clasificadores Haar). La ruta a estos archivos puede variar dependiendo del sistema operativo.

La detección de objetos en una escena con Haarcascades se realiza utilizando el nodo disparity_track.py. Este documento escrito en Phyton requiere simplemente que se le proporcione la ruta del archivo.xml con los datos del clasificador en cascada Haar. En el script, dicha línea es la siguiente:

```
cascade = cv2.CascadeClassifier('/usr/share/opencv4/haarcascades/
    haarcascade_frontalface_alt.xml')
```

La inicialización del nodo de detección por Haarcascades se especifica en el documento launch con la siguiente línea de código:

```
<node name="tracker_node" pkg="stereo_color_tracker" type="
    disparity_track.py" output="screen">
```

Algo a señalar sobre este nodo es que el script del mismo incluye una serie de instrucciones para enmarcar el objeto que el algoritmo esta detectando en su momento.

2.3.3. Identificación de objetos con red neuronal

La paquetería para la detección de objetos con la ayuda de una red neuronal se llama dnn_detect y puede hallarse en el siguiente repositorio: *https : //github.com/UbiquityRobotics/dnn_detect*.

Esta paquetería implementa la red neuronal convolucional llamada YOLO. Se trata de red neuronal para la detección rápida y precisa de objetos creada por Joseph Redmond como principal autor [36], la cual predice los objetos presentes en la imagen y sus cuadros delimitadores de una sola pasada.

A diferencia de las propuestas basadas en regiones donde necesitamos extraer características y clasificarlas con diferentes redes, YOLO realiza tanto la extracción de características como la clasificación de objetos y la predicción del cuadro delimitador utilizando una sola red. Para hacer las predicciones, a diferencia de los métodos anteriores, este sistema consume la imagen completa, en lugar de regiones de la misma. Esto hace que se limiten los errores a la hora de reconocer las clases de objetos que hay en la imagen.

Para utilizar dnn_detect, primero se deben descargar los archivos del repositorio correspondiente con:

```
1     sudo apt install ros-kinetic-dnn-detect
```

Descargada y ubicada la paquetería en el espacio de trabajo Catkin, podemos ejecutar la misma y sus nodos con la siguiente instrucción en el archivo launch:

```
1 <node pkg="dnn_detect" name="left_detect" type="dnn_detect" output=
      "screen" respawn="false">
2 <node name="tracker_node" pkg="stereo_color_tracker" type="
      nn_v1_track.py" output="screen">
3      </node>
```

El algoritmo en tiempo real se puede utilizar en casi todos los escenarios, independientemente de las condiciones de la fuente, la iluminación y la resolución de la imagen, la pose, la posición y las expresiones faciales. En pocas palabras, dnn_detect es muy precisa y fácil de usar.

2.4. Cálculo de disparidad

En cada una de las paqueterías de deteccoón de objetos se incluyo un script que contiene las instrucciones para el tratamiento de las imágenes y la obtención de la disparidad. El archivo en cuestión recive el nombre de disparity_track.py y en el mismo las líneas de código de mayor relevancia son las siguientes:

```
1     ##Funcion para calculo de profundidad
```

```
2
3     def Calculations(self, dxr, dxl):
4         disparity= dxl-dxr
5         try:
6             z = (self.base*self.focal/disparity)*.0002645833
7             if z < 0:
8                 print('Error: Profundidad negativa')
9                 return None
10            else:
11                return z
12        except:
13            print('Error: No hay datos')
14            return None
```

Note que estas líneas de código contienen la ecuación para determinar la profundidad a partir de la disparidad, misma que esta dada por las coordenadas del centroide de la región del objeto detectado en cada imagen. Adicionalmente, la profundidad z es multiplicada por un factor de 0002645833. Este factor permite obtener la profundidad en centḿetros, pues recordemos que las disparidad esta dada en términos de píxeles de la imagen.

Capítulo 3

Resultados y discusión

En este capítulo se presentan las estimaciones de profundidad obtenidas durante los experimentos. Recordemos que las metodologías para la detección de objetos son los encargados de ubicar un objeto particular en la imágen, no obstante la ecuación para determinar la profundidad es la misma siempre (ecuación (1.9)). Para obtener estos resultados, el par binocular se colocó en una posición fija y se orientó hacia un espacio que permitiera acercarse y/o alejarse del mismo. En la práctica es el arreglo binocular el que se desplaza, puesto que suele montarse en un vehículo. No obstante para nuestros experimentos fue el objeto en cuestión el que se desplazó, esto por comodidad dado que mover el arreglo binocular implicaría mover la electrónica adicional (arduino, ordenador, etc.). Se colocaron marcas en el suelo de la escena cada 50 centímetros. Por otro lado, el objeto a detectar cambió dependiendo del método de detección de objetos utilizado.

Con segmentación de color, se experimentó con objetos de color uniforme, en particular, con objetos de tonalidades azules. Para Haarcascades el objeto en cuestión fue un rostro, así como otros objetos que se asemejaran. Por último, con la Red Neuronal, se probó con diferentes objetos: personas, gatos, perros, sillas y botellas.

Durante los experimentos se obtuvieron lecturas de profundidad arrojadas directamente por la expresión (1.9), es decir, se obtuvieron lecturas a partir de las coordenadas x (izquierda y derecha, las cuales son del centroide de la región detectada por el algoritmo en cuestión, mismo que es calculado por OpenCv), la distancia focal f, la línea base b (11cm) y la disparidad d. A estas estimaciones las llamamos *Datos originales*. Sin embargo es bien sabido que una práctica muy común al momento de realizar mediciones es implementar algún tipo de filtro que permita obtener lecturas más suaves. Así que decidimos implementar una media móvil sobre los datos originales, a estos nuevos resultados los llamamos

Datos suavizados.

3.1. Resultados segmentación por color

Para el experimento con el método de segmentación de color, el objeto a detectar se colocó inicialmente a una distancia de 50 cm del arreglo binocular. Se tomó esta decisión dado que a esa distancia el algoritmo comenzó a proporcionar datos de profundidad estimada. Posteriormente mientras el algoritmo continuaba en ejecución el objeto se desplazó un metro, se mantuvo ahí por alrededor de 10 segundos para posteriormente desplazarse nuevamente un metro, y así sucesivamente (50 cm, 150 cm, 250 cm, 350 cm, 450 cm). Las lecturas concluyeron cuando el objeto se ubicó a 5 metros del arreglo binocular, debido a que el espacio disponible para realizar las mediciones era de 5 metros.

La Figura 3.1 muestra las imágenes capturadas por el par binocular y el procesamiento que estas reciben por la segmentación de color. Notemos que las imágenes muestran en color negro todo aquello en la escena que no coincide con el color buscado (en este caso el color azul); y al mismo tiempo las imágenes muestran el centroide de la región detectada (punto blanco). Recordemos que el centroide nos proporciona información sobre las coordenadas x del objeto en la imagen correspondiente, información que posteriormente es utilizada para estimar la profundidad a la que se encuentra el objeto detectado. Por conveniencia del usuario, el algoritmo también despliega en la imagen el valor de profundidad estimado en el instante correspondiente.

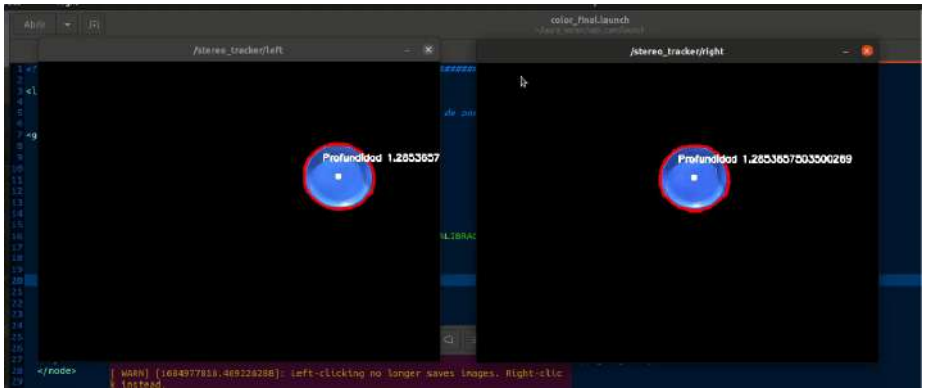
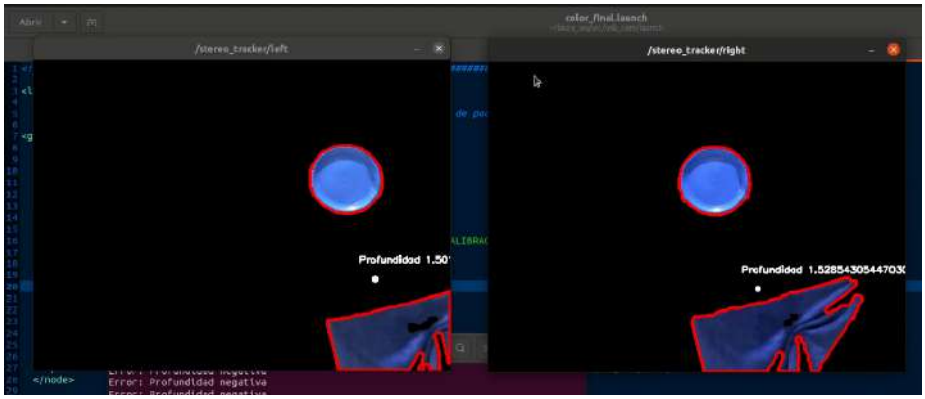


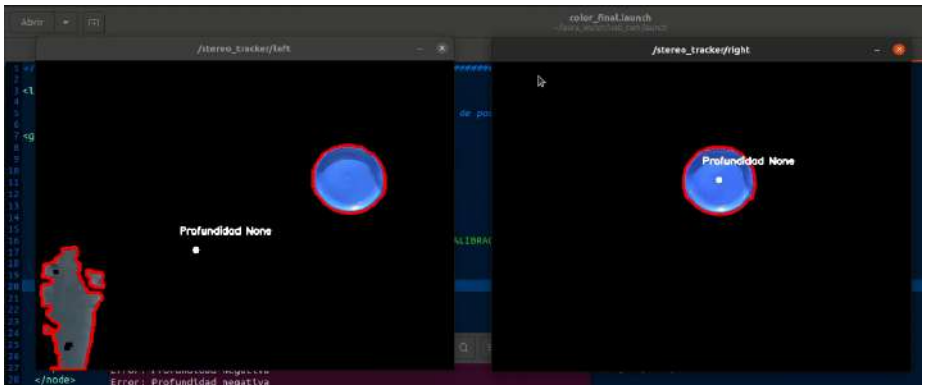
Figura 3.1: Caso ideal: Detección de un objeto por segmentación de color y profundidad estimada.

Un inconveniente de la segmentación de color, es que se confunde con facilidad cuando múltiples objetos de la misma tonalidad de color aparecen en el campo de visión binocular. En tal escenario, al detectarse nuevas regiones

que cumplen con estas características, el centroide obtenido por el algoritmo se desplaza, dando como resultado estimaciones de profundidad erróneas. Por ejemplo, en la Figura 3.2a, al aparecer un segundo objeto de color azul en la escena (región no uniforme), el centroide calculado se desplaza a una posición fuera de la región de interés (región circular).



(a)



(b)

Figura 3.2: Casos en donde el algoritmo falla. a) Desplazamiento de centroide debido a múltiples regiones del mismo color y b) caso de oclusión donde la segunda región de color no es detectada por la cámara derecha.

Por otro lado, debemos tener en cuenta que la segmentación de color se ve influenciada por otros factores como los cambios de iluminacón, texturas de los objetos y oclusiones, mismas que generan confusiones al algoritmo.

En este experimento en particular, se decidió detectar objetos con tonalidades azules, dado que de acuerdo a algunas fuentes, la detección suele ser mas sencilla cuando se tienen señales de colores brillantes, como el azul o el rojo [37]. Si bien se probó con distintos colores, fue el color azul el que presentó los mejores

resultados de segmentación con respecto a los cambios de iluminación.

A su vez, observando la Figura 3.2b, podemos notar que no se muestra información de profundidad. Esto se debe a una oclusión, es decir, la cámara izquierda detecta o ve una región de color que no es observada por la cámara derecha. En particular, debido al desplazamiento del centroide calculado en la imagen izquierda, la disparidad toma un valor negativo, mismo que por supuesto no es posible, tal que el algoritmo decide en su lugar regresar el valor *None*.

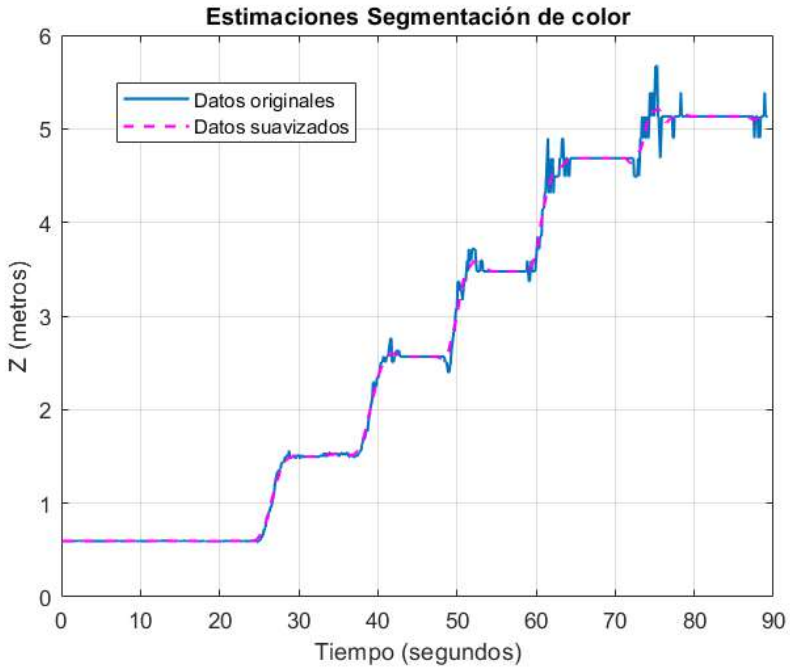


Figura 3.3: Estimaciones de profundidad de un objeto detectado con el algoritmo de segmentación de color.

En la Figura 3.3 podemos observar las estimaciones obtenidas durante este experimento. Notemos que los datos originales presentan pocas oscilaciones, es decir, las lecturas son estables. Al aplicar la media móvil las estimaciones son totalmente suaves. Recordemos que el objeto es desplazado después de unos segundos, este desplazamiento explicaría los picos que presenta la curva azul. Debemos mencionar que las condiciones bajo las que se obtuvieron estas mediciones no presentaron objetos en la escena que pudieran confundir al algoritmo.

Tabla3.1

Valor real (m)	Valor estimado (m)	Error absoluto (m)	Error relativo (%)
0.50	0.596	0.096	19.196
1.50	1.525	0.025	1.692
2.50	2.566	0.066	2.654
3.50	3.477	0.033	0.656
4.50	4.686	0.186	4.142
5	5.132	0.132	2.654

En la tabla 3.1 podemos observar el error absoluto y el error relativo de las estimaciones obtenidas, recordando que el error relativo se calcula como:

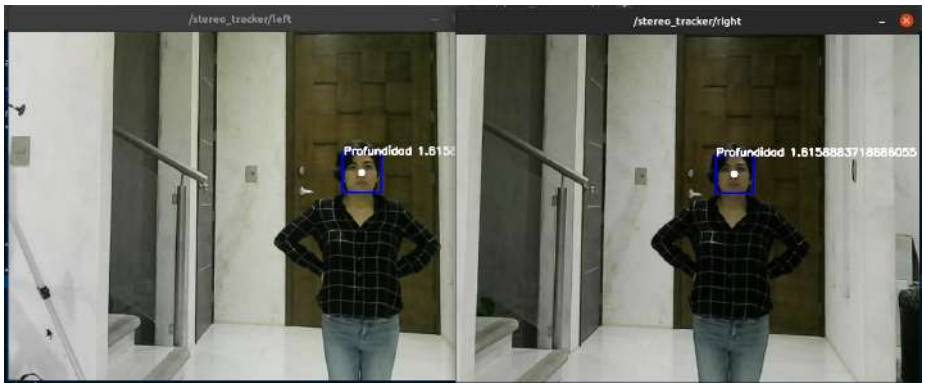
$$\text{Error Relativo} = \frac{\text{Error Absoluto}}{|r|}\frac{|r-z|}{|r|} \tag{3.1}$$

Donde r es el valor real y z es el valor estimado. Notemos que el error absoluto para las primeras cuatro mediciones es menor a 10 cm, mientras que el error absoluto para 4.5 m y 5 m oscila alrededor de los 15 cm. No obstante, al calcular el error relativo encontramos que para todas las estimaciones con excepción del primer valor, este es menor al 5 %. Por lo tanto podemos decir que estas estimaciones son apropiadas para una posible aplicación. Es interesante observar que la lectura obtenida con el arreglo binocular, separado 50 cm del objeto, presenta un error relativo casi del 20 %. Si bien el error absoluto en este caso es menor que por ejemplo en las últimas dos estimaciones, debido a la proporción de la escala el error relativo es bastante alto. El error observado en este estudio se debe en mayor contribución al dispositivo de visión binocular. Como se menciona, una de las limitaciones de este estudio es el equipo utilizado para procesar la información que se recibe del arreglo.

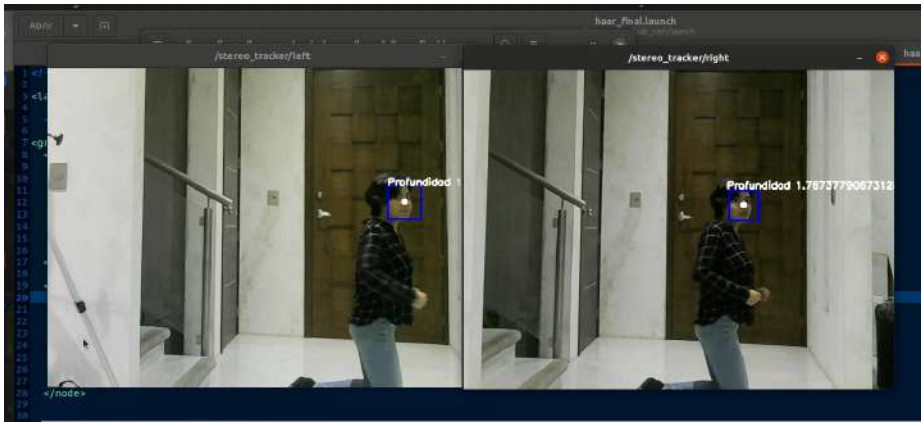
3.2. Resultado de Haarcascades

Con el algoritmo de Haarcascades se realizaron diferentes pruebas para la detección de rostros. En estas se tomaron en cuenta diferentes orientaciones del rostro. En cada experimento se ubicó el objeto a una distancia de 1 metro del par binocular. Se mantuvo esa posición durante unos segundos hasta que se estabilizara la estimación de profundidad. El procedimiento fue repetido desplazando el objeto un metro más hasta que el mismo dejó de ser detectado por el par binocular.

A diferencia del experimento anterior, aquí (Figura3.4)se muestra una escena que corresponde al campo de visión de cada cámara y se enmarca el objeto



(a)



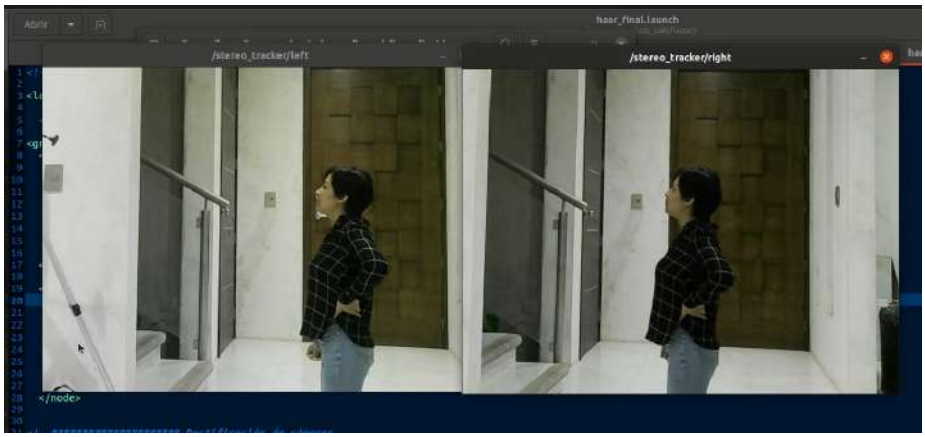
(b)

Figura 3.4: Detección de rostros a partir del modelo en cascada. a) Detección de rostro visto frontalmente y b) detección de rostro visto parcialmente.

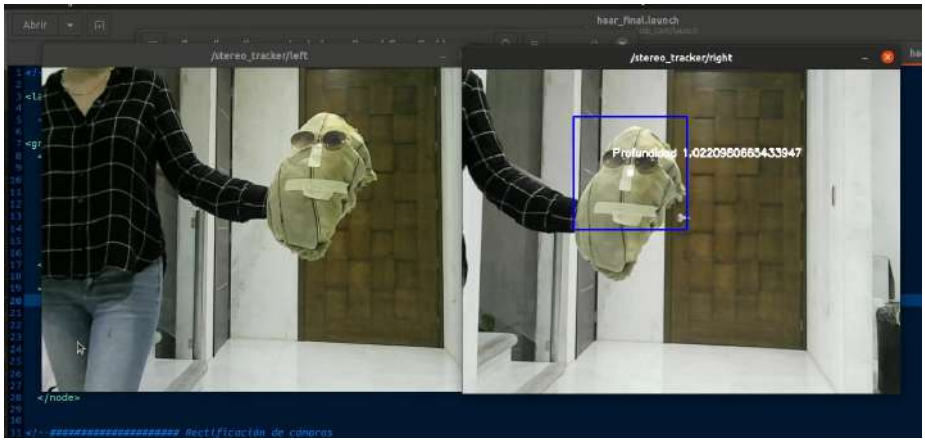
para el cual ha sido entrenado para detectar. En el caso de los resultados que aquí se muestran, el objeto a detectar es un rostro. No obstante, se realizaron pruebas con diferentes objetos, puesto que OpenCv cuenta con herramientas preentrenadas para la detección de ojos, cuerpos, etc.

En las Figuras 3.4b y 3.4a, el algoritmo localiza el rostro en la imagen, lo enmarca con un rectángulo, se calcula el centroide de esta región enmarcada y en base a esta información se estima la profundidad. Notemos que adicionalmente, al igual que en el experimento anterior, se muestra la lectura de profundidad en el instante correspondiente.

Una desventaja de Haarcascades es que se puede confundir con facilidad dado que la información que utiliza este algoritmo para detectar objetos particulares es relativamente escasa. Si bien esta característica hace que Haarcascades sea



(a)



(b)

Figura 3.5: Errores de detección generados por Haarcascade. a) generación de un falso negativo y b) generación de un falso positivo.

relativamente poco costoso de implementar, también da como resultado lo ya mencionado. Por ejemplo, se observa que un rostro no es identificado cuando se ve de perfil, tal y como se muestra en la Figura 3.5a.

En general, para este algoritmo pueden suceder dos tipos de errores: 1) rostros que son pasados por alto debido a que la información visual recibida por el arreglo no es suficiente. 2) Por otro lado, falsos positivos en los que considera que una región de la imagen es un rostro, pero no lo es. Este resultado puede ser debido a que para reconocer un rostro existen grandes variaciones a tomar en cuenta por el modelo, como lo es, la cantidad de luz o ausencia de esta, los accesorios (lentes,gorras, aretes, etc.), el maquillaje, los gestos, etc. Como se

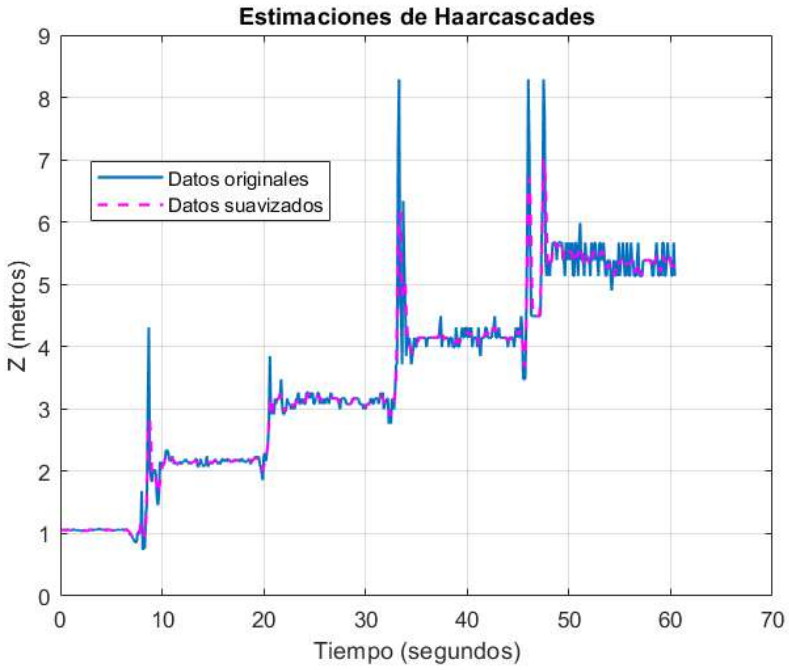


Figura 3.6: Estimaciones de profundidad de un objeto detectado con el algoritmo de Haarcascades. Note que la posición del objeto respecto al arreglo binocular varia.

puede observar en la Figura 3.5b, donde se logró confundir al algoritmo a base de simular facciones simples de un rostro en un cojín.

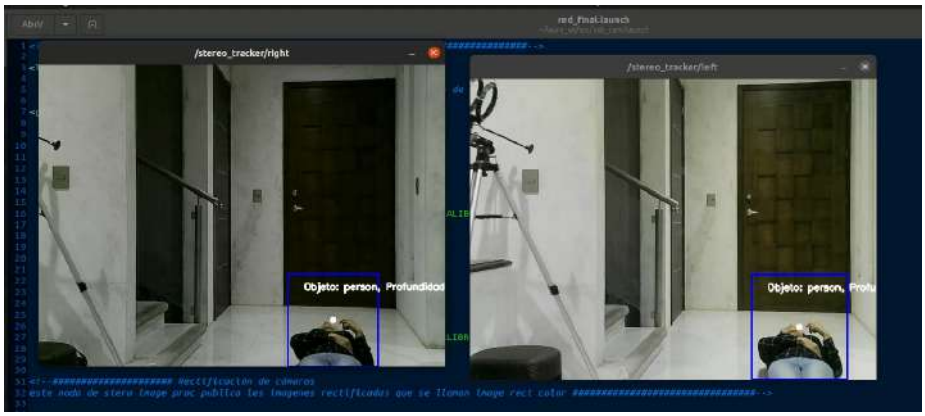
En la Figura 3.6, la línea azul muestra las estimaciones de profundidad obtenidas por el algoritmo. En esta, se observan variaciones en la lectura. Estas variaciones aumentan y tardan más en estabilizarse conforme aumenta la distancia que hay entre el objeto y el arreglo. Estas variaciones en la curva azul pueden explicarse por la tendencia del algoritmo a generar falsos positivos o negativos. Otra explicación tentativa es su sensibilidad a los cambios de iluminación.

Tabla3.2

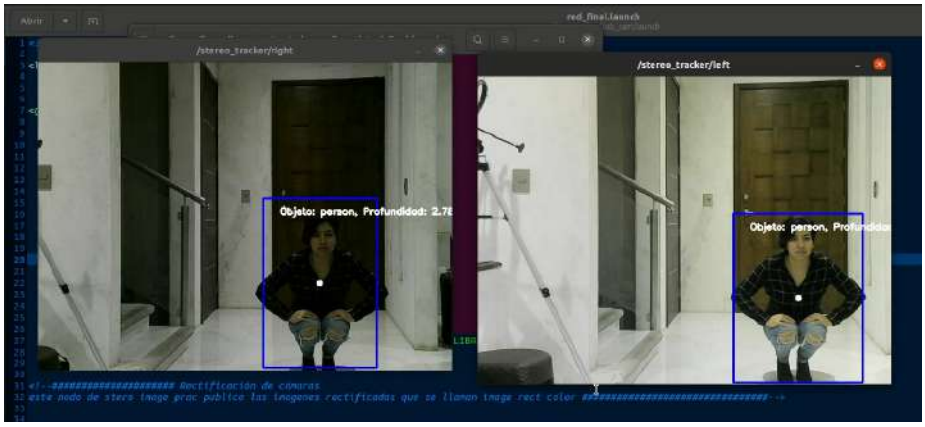
Valor real (m)	Valor estimado (m)	Error Absoluto (m)	Error Relativo (%)
1	1.056	0.056	5.579
2	2.145	0.145	7.259
3	3.148	0.148	4.965
4	4.228	0.228	5.714
5	5.332	0.332	6.6394

En la tabla 3.2 se presentan los errores absolutos y errores relativos obtenidos con Haarcascades. Notemos que prácticamente todos los errores superan el 5 %, es decir, si bien este algoritmo es computacionalmente poco costoso, sus estimaciones no son fiables.

3.3. Resultados con Red Neuronal



(a)

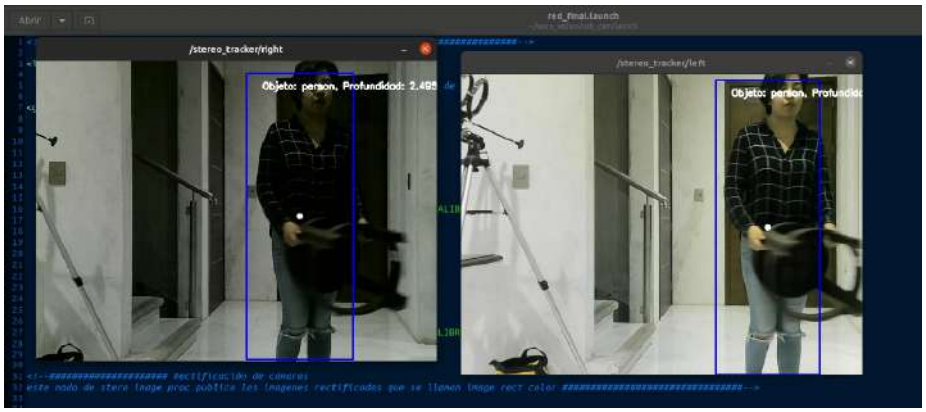


(b)

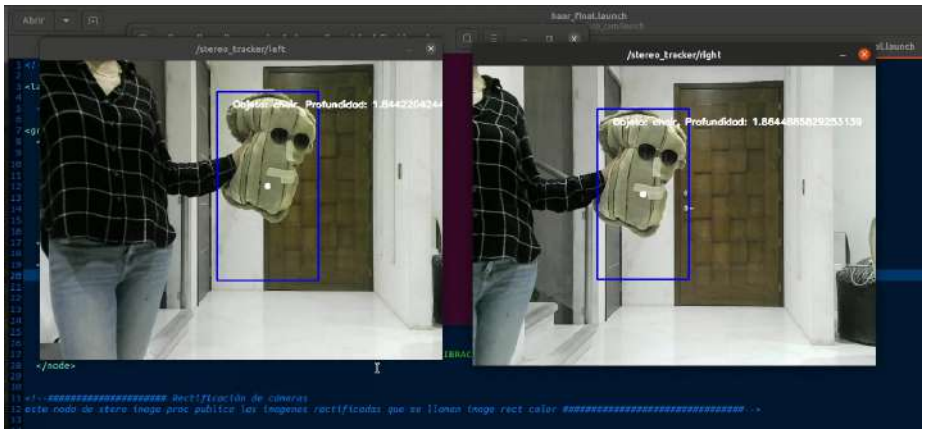
Figura 3.7: Detección de una persona sin importar su posicionés haciendo uso de una red neuronal

Las estimaciones de profundidad obtenidas por la red neuronal pueden observarse en la Figura 3.7. Para las pruebas con la red neuronal, el objeto a detectar se ubicó inicialmente a un metro y posteriormente se fue desplazando un metro adicional. El procedimiento fue repetido hasta que el objeto dejó de

ser detectado por el par binocular. En las Figuras 3.7a y 3.7b se muestra como la red neuronal es capaz de detectar a una persona sin importar el ángulo o la posición de esta. En otras palabras, la red neuronal se confunde con mayor dificultad respecto a los algoritmos anteriores.



(a)



(b)

Figura 3.8: Detección de una persona y un cojin usando red neuronal

Un aspecto importante a tomar en cuenta con la red neuronal es que esta es capaz de operar sin retardos con imágenes de 640X480 píxeles a 10fps, a diferencia de los algoritmos anteriores, los cuales después de cierto tiempo comenzaron a mostrar retardos pese a funcionar con los mismos fps.

Los resultados con la red neuronal muestran que este modelo difícilmente se confunde, tal y como se muestra en la Figura 3.7. Observe además que la red neuronal es capaz de detectar personas que son observadas parcialmente, como se aprecia en la Figura 3.7a. En todas las figuras puede notarse que el

área enmarcada indica la detección de un cuerpo/persona, tambieén se muestra el centroide de la región enmarcada y la etiqueta del valor de su profundidad.

Otra ventaja que presenta el uso de redes neuronales con respecto a los algoritmos anteriores, es que es capaz de identificar personas aún cuando hay objetos ajenos interponiéndose entre la persona y el par binocular. Eso lo podemos observar en la Figura 3.8a, donde a pesar que el individuo sostiene una silla, la red neuronal es capaz de detectarlo. Por otro lado en la Figura3.8, notamos que la red YOLO detecta como un cojín aún cuando estes intenta simular facciones humanas. Recordemos que este mismo objeto es detectado por Haarcascades como un rostro.

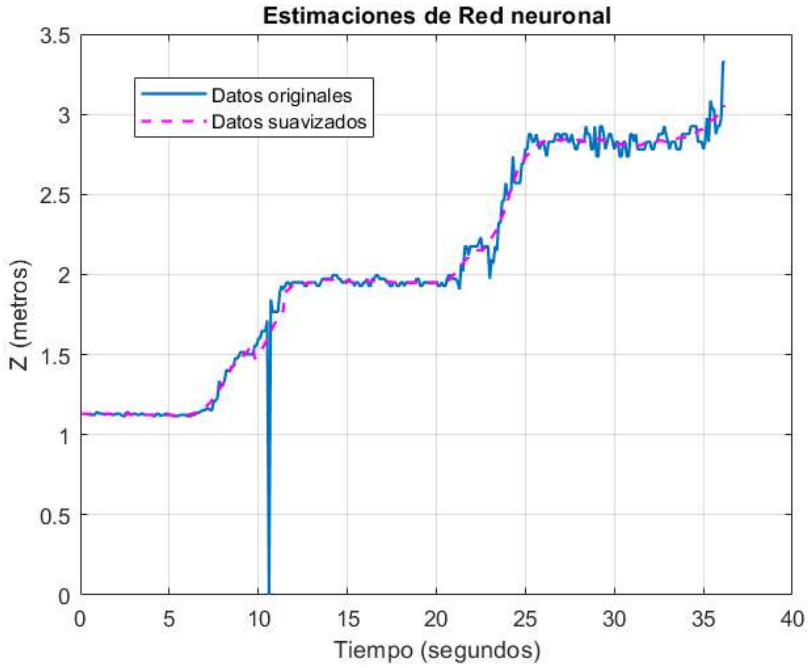


Figura 3.9: Estimaciones de profundidad de un objeto detectado con una red neuronal convolucional (pre-entrenada). Note que la posición del objeto respecto al arreglo binocular varia.

En la Figura 3.9 podemos observar que el alcance de detección de este algoritmo es sólo de 3 metros, a diferencia de los anteriores. No obstante, la red neuronal no tiende a generar falsas detecciones conforme los objetos van alejándose del par binocular. Notemos que las estimaciones de profundidad obtenidas con este algoritmo aun presentan variaciones, aunque las lecturas tienden a estabilizarse con mayor rapidez que los algoritmos anteriores.

Tabla3.3

Valor real (m)	Valor estimado (m)	Error Absoluto (m)	Error Relativo (%)
1	1.122	0.122	12.278
2	1.958	0.042	2.080
3	2.820	0.18	5.972

En la tabla 3.3 podemos encontrar los errores absolutos y relativos correspondientes. Notemos que solo la lectura para 2 metros es inferior al 5 %, de tal manera que si bien la red neuronal no confunde objetos, en general sus estimaciones de profundidad no son fiables.

3.4. Discusión de resultados

Los resultados obtenidos evidencian las limitaciones y ventajas de los algoritmos de detección de objetos. Los resultados de este estudio indican que en general, los errores y las variaciones de las estimaciones de profundidad son generados por condiciones de iluminación y/o falsos positivos (falsos negativos) arrojados por los algoritmos de detección 3.10.

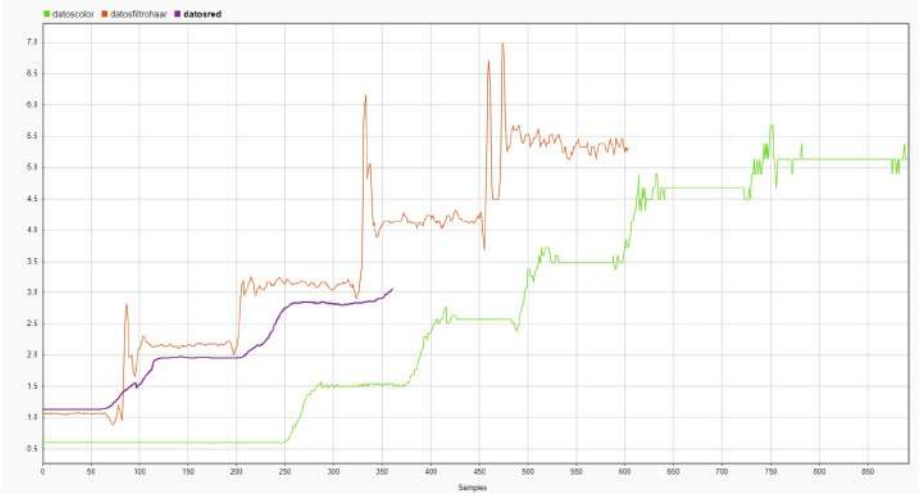


Figura 3.10: Comparación de los resultados obtenidos con las técnicas de detección de objetos utilizadas .

La segmentación de color mostró ser la más fiable respecto a las estimaciones de profundidad dado que el error relativo de estas está por debajo del 5 %, debido a que en este algoritmo la región detectada variaba muy poco dando como

resultado un centroide calculado más estable. No obstante, también observamos que es un algoritmo muy fácil de confundir cuando cambian las condiciones de iluminación y/o aparecen en la escena múltiples objetos del mismo color.

Por otro lado, los algoritmos basados en clasificadores presentan estimaciones de profundidad con un error relativo por encima del 5 %, es decir, estas estimaciones no son aptas para implementaciones prácticas. Podemos atribuir estas estimaciones tan poco fiables al hecho de que Haarcascades tiende a generar muchos falsos positivos o falsos negativos. Si bien, entrenar e implementar un clasificador Haarcascades es computacionalmente poco costoso, esto no justifica su imlementación para estimar profundidad debido al alto error relativo.

Con la red neuronal pudimos observar que la misma oscila muy poco dado que no se confunde con facilidad. No obstante también notamos que no es capaz de detectar objetos más allá de los 3 metros y prácticamente todos sus errores relativos superan el 5 %. En otras palabras, la red neuronal es la más eficiente para detectar objetos pero no así para estimar la profundidad.

Conclusiones

El estudio presentado en este trabajo de tesis considera la integración de tres algoritmos diferentes de detección de objetos: algoritmo de segmentación por color, redes neuronales convolucionales y extracción en cascada de características Haar. Se observó que cada técnica de detección presenta ventajas y desventajas. Desde una mayor sensibilidad a las condiciones del entorno, pasando por una mayor o menor tasa de falsos positivos y falsos negativos, hasta la capacidad de procesar un mayor número de imágenes. Por ejemplo, el beneficio del entrenamiento e implementación de redes neuronales convolucionales fue una baja tasa de falsos positivos y negativos a costa de estimaciones menos precisas. Por otro lado, la técnica de segmentación de color arroja datos mas precisos pero se confunde con facilidad ante los cambios de iluminación y la presencia de múltiples objetos con tonalidades de color similares. Por tanto, en este trabajo se concluye que para futuras aplicaciones practicas, la mejor idea es una combinación de estas técnicas.

Tras realizar los experimentos podemos concluir que no hay una metodología superior dado que cada una presenta ventajas y desventajas. En la detección de objetos por color se observó principalmente el inconveniente generado por la superposición de regiones de tonalidades similares. Este efecto generó gran confusión en el algoritmo y representa un gran incoveniente, sobre todo si se considera una posible aplicación en un sistema de navegación autónomo. No obstante, esta técnica exhibe ciertas ventajas, tales como la velocidad de detección y la exactitud en el cálculo de la distancia. Los resultados muestran estimaciones con pocas variaciones y muy precisas; como se observó en la tabla 3.1 el cálculo del error relativo, se encontró que para todas las estimaciones con excepci´on del primer valor, este es menor al 5 %. Por lo tanto podemos decir que estas estimaciones son apropiadas para una posible aplicación. Superando a las otras técnicas. Por otro lado, con Haarcascades se observó muchas limitaciones debido a que tiende a generar falsos positivos/negativos, tal y como mostramos con el cojín con lentes que asemeja un rostro humano, así como estimaciones de profundidad tabla 3.2 Notemos que prácticamente todos los errores superan el 5 %, es decir, si bien este algoritmo es computacionalmente poco costoso, sus estimaciones no son fiables. Por su parte la Red Neuronal es el algoritmo que

se confunde menos, es decir, es el mejor para detectar objetos. Si bien tampoco proporciona estimaciones de profundidad fiables, como pudimos observar en la tabla 3.3. Notemos que solo la lectura para 2 metros es inferior al 5 %, de tal manera que si bien la red neuronal no confunde objetos, en general sus estimaciones de profundidad no son fiables.

Algo que podemos concluir es que idealmente una futura posible aplicación debería incluir al menos dos de las técnicas utilizadas. En particular creemos que implementar segmentación de color junto con una Red Neuronal resultaría en estimaciones más precisas y a la vez en un algoritmo que se confunda con mayor dificultad. En otras palabras, el posible algoritmo a implementar consistiría en 1) detectar un objeto con la Red Neuronal y aislar la región correspondiente y 2) posteriormente segmentar por color dicha región. Esto nos permitiría evitar que el algoritmo se confundiera ante la presencia de múltiples objetos del mismo color y los cambios de iluminación.

Tengamos en cuenta que si bien en este trabajo observamos los efectos la iluminación, los obstáculos, las oclusiones y demás, estos no fueron analizados con mayor rigurosidad debido al alcance de este trabajo de tesis. Recordemos que el objetivo de este estudio se limitó a familiarizarce con algunas de las técnicas de visión artificial que se están implementando para impulsar o desarrollar la navegación de los sistemas robóticos. Por lo tanto estudiar los efectos arriba mencionados podria ser un posible trabajo a futuro. Por ejemplo, la iluminación es fundamental en un sistema de visión dado que la luz en la escena favorece la fomación de la imagen y los posteriores procesos que se realicen sobre ella, por lo que en un trabajo a futuro se podrían estudiar los tipos de iluminación artificial.

A su vez, en investigaciones futuras, existe la posibilidad de adaptar un sistema de visión artificial con un sistema submarino para la detección de cuerpos. Con el fin de identificar condiciones para su implementación en diferentes entornos. Para lo cual también y de acuerdo a los resultados observados con respecto a la detección de objetos; indican que vale la pena, de ser posible, invertir en el entrenamiento de una Red Neuronal dado que se confunde con muchísima menos dificultad que Haarcascades y a su vez es capaz de operar en tiempo real. También debemos senãlar que si bien fuimos capaces de implementar estos algoritmos a 10fps, si dispusieramos de un ordenador más sofisticado podríamos incrementar el número de frames y obtener así lecturas más continuas.

Para concluir, tenemos que los resultados obtenidos refuerzan la idea de que la integración de diferentes métodos de adquisición de información en un sistema de visión es deseada. La integración de múltiples algoritmos de detección permitirían obtener información más precisa que facilite la toma de decisiones. Es otras palabras, la implementación en sistemas tales como vehículos de navegación autónoma o sistemas de supervisión idealmente involucrará distintos métodos de detección de objetos.

Para este trabajo se realizó una revisión de diferentes algoritmos de detección de objetos y se implementó un arreglo de visión artificial binocular. Visión artificial es una disciplina que inspira diferentes y variadas aplicaciones que son sujeto de estudio. En la Benemérita Universidad Autónoma de Puebla, que, si bien, los trabajos hallados corresponden a la facultad de computación y estos en su mayoria se centran particularmente en la evaluació de los algoritmos como en [38], y [39]. Este trabajo, se propone con la intención de introducir el estudio de esta disciplina en la facultad de Ciencias de la Electrónica.

Apéndice

3.5. Productos del trabajo de tesis

- **LXVI Congreso Nacional de Física**

Figura 3.11: Constancia de participación en el LXVI CNF

- **22 Congreso Nacional de mecatrónica.**

Figura 3.12: Constancia de participación en el 22 congreso nacional de mecatrónica

3.6. Códigos

```
1
2  <launch>
3   <group ns="stereo">
4    <node name="left" pkg="usb_cam" type="usb_cam_node" output="
       screen" >
5      <param name="video_device" value="/dev/video2" />
6      <param name="image_width" value="640" />
7      <param name="image_height" value="480" />
8      <param name="pixel_format" value="yuyv"/>
9      <param name="camera_frame_id" value="left_cam" />
10     <param name="io_method" value="mmap"/>
11     <param name="framerate" value="10"/>
12   </node>
13   <node name="image_view_1" pkg="image_view" type="image_view"
       respawn="false" output="screen">
14     <remap from="image" to="/stereo/left/image_raw"/>
15     <param name="autosize" value="true" />
16   </node>
17
18
```

```
  <node name="right" pkg="usb_cam" type="usb_cam_node" output="
    screen" >
    <param name="video_device" value="/dev/video4" />
    <param name="image_width" value="640" />
    <param name="image_height" value="480" />
    <param name="pixel_format" value="yuyv" />
    <param name="camera_frame_id" value="right_cam" />
    <param name="io_method" value="mmap"/>
    <param name="framerate" value="10"/>
  </node>
  <node name="image_view_2" pkg="image_view" type="image_view"
    respawn="false" output="screen">
    <remap from="image" to="/stereo/right/image_raw"/>
    <param name="autosize" value="true" />
  </node>


 </group>









</launch>
```

Listing 3.1: Launch para iniciar dos camaras

```

#include <AccelStepper.h>
#include <MultiStepper.h>

// Define stepper motor connections and steps per revolution:
#define dirPin 2
#define stepPin 3
#define stepsPerRevolution 200

void setup() {
  // Declare pins as output:
  pinMode(stepPin, OUTPUT);
  pinMode(dirPin, OUTPUT);
}

void loop() {
  // Set the spinning direction clockwise:
  digitalWrite(dirPin, LOW);

  // Spin the stepper motor 1 revolution slowly:
  for (int i = 0; i < 16 * stepsPerRevolution; i++) {
    // These four lines result in 1 step:
    digitalWrite(stepPin, HIGH);
    delayMicroseconds(1500);
    digitalWrite(stepPin, LOW);
    delayMicroseconds(1500);
  }

  delay(800);
   // Set the spinning direction clockwise:
  digitalWrite(dirPin, HIGH);
```

```
32
33   // Spin the stepper motor 1 revolution slowly:
34   for (int i = 0; i < 7 * stepsPerRevolution; i++) {
35     // These four lines result in 1 step:
36     digitalWrite(stepPin, HIGH);
37     delayMicroseconds(1500);
38     digitalWrite(stepPin, LOW);
39     delayMicroseconds(1500);
40   }
41   delay(2000);
42  // Spin the stepper motor 1 revolution slowly:
43   for (int i = 0; i < 9 * stepsPerRevolution; i++) {
44     // These four lines result in 1 step:
45     digitalWrite(stepPin, HIGH);
46     delayMicroseconds(1500);
47     digitalWrite(stepPin, LOW);
48     delayMicroseconds(1500);
49   }
50   delay(1000);
51 }
```

Listing 3.2: Programa que controla la dirección del movimiento del motor

```
1
2 <!-- Launch principal para estimaci n de profundidad -->
3
4 <launch>
5
6   <!-- Inicializaci n de c maras USB haciendo uso de paqueter a
      usb_cam -->
7
8 <group ns="stereo">
9   <node name="left" pkg="usb_cam" type="usb_cam_node" output="
      screen" >
10    <param name="video_device" value="/dev/video2" />
11    <param name="image_width" value="640" />
12    <param name="image_height" value="480" />
13    <param name="framerate" value="10" />
14    <param name="pixel_format" value="yuyv" />
15    <param name="camera_frame_id" value="left" />
16    <param name="io_method" value="mmap"/>
17     <param name="camera_info_url" value="file:///home/laura/
      Escritorio/CALIBRACIONMAYO/Lb18cm_2105/left.yaml"/>
18  </node>
19
20  <node name="right" pkg="usb_cam" type="usb_cam_node" output="
      screen" >
21    <param name="video_device" value="/dev/video4" />
22    <param name="image_width" value="640" />
23    <param name="image_height" value="480" />
24    <param name="framerate" value="10" />
25    <param name="pixel_format" value="yuyv" />
26    <param name="camera_frame_id" value="right" />
27    <param name="io_method" value="mmap"/>
28    <param name="camera_info_url" value="file:///home/laura/
      Escritorio/CALIBRACIONMAYO/Lb18cm_2105/right.yaml"/>
29  </node>
30
```

```
<!--Rectificaci n de c maras este nodo de stero image proc
    publica las imagenes rectificadas que se llaman image rect
    color -->

<node name="stereo_image_proc" pkg="stereo_image_proc" type="
    stereo_image_proc" output="screen" >
<param name="queue_size" value="10" />
<param name="approximate_sync" value="True" />
<remap from="/stereo/left/image" to="/stereo/left/image_raw"/>
<remap from="/stereo/right/image" to="/stereo/right/image_raw"/>
</node>

</group>

<!--##Identificaci n de objeto por color ##-->

<node name="tracker_node" pkg="stereo_color_tracker" type="
    track_color.py" output="screen">
<remap from="/stereo/left/image_raw" to="/stereo/left/
    image_rect_color"/>
<remap from="/stereo/right/image_raw" to="/stereo/right/
    image_rect_color"/>
</node>

<!-- Visualizaci n de im genes capturadas por las c maras usb
    haciendo uso de image_view -->

  <node name="image_view_left" pkg="image_view" type="image_view"
    respawn="false" output="screen">
    <remap from="image" to="/stereo_tracker/left"/>
    <param name="autosize" value="true" />
  </node>


  <node name="image_view_right" pkg="image_view" type="image_view"
    respawn="false" output="screen">
    <remap from="image" to="/stereo_tracker/right"/>
    <param name="autosize" value="true" />
  </node>
</launch>
```

Listing 3.3: Launch que realiza las tareas de detección de objetos y calcula su profundidad con modelo de segmentación de color

```
<!-- Launch principal para estimaci n de profundidad -->

<launch>

  <!-- Inicializaci n de c maras USB haciendo uso de paqueter a
    usb_cam -->

<group ns="stereo">
  <node name="left" pkg="usb_cam" type="usb_cam_node" output="
    screen" >
    <param name="video_device" value="/dev/video2" />
    <param name="image_width" value="640" />
    <param name="image_height" value="480" />
```

```
12      <param name="framerate" value="10" />
13      <param name="pixel_format" value="yuyv" />
14      <param name="camera_frame_id" value="left" />
15      <param name="io_method" value="mmap"/>
16       <param name="camera_info_url" value="file:///home/laura/
        Escritorio/CALIBRACIONMAYO/Lb18cm_2105/left.yaml"/>
17    </node>
18
19    <node name="right" pkg="usb_cam" type="usb_cam_node" output="
        screen" >
20      <param name="video_device" value="/dev/video4" />
21      <param name="image_width" value="640" />
22      <param name="image_height" value="480" />
23      <param name="framerate" value="10" />
24      <param name="pixel_format" value="yuyv" />
25      <param name="camera_frame_id" value="right" />
26      <param name="io_method" value="mmap"/>
27      <param name="camera_info_url" value="file:///home/laura/
        Escritorio/CALIBRACIONMAYO/Lb18cm_2105/right.yaml"/>
28    </node>
29
30
31  <!--###################### Rectificaci n de c maras
32  este nodo de stero image proc publica las imagenes rectificadas que
        se llaman image rect color ################################--->
33
34
35  <node name="stereo_image_proc" pkg="stereo_image_proc" type="
        stereo_image_proc" output="screen" >
36  <param name="queue_size" value="10" />
37  <param name="approximate_sync" value="True" />
38  <remap from="/stereo/left/image" to="/stereo/left/image_raw"/>
39  <remap from="/stereo/right/image" to="/stereo/right/image_raw"/>
40  </node>
41
42  </group>
43
44
45
46  <!-- ###################### Identificaci n de objetos y c lculo
        de disparidad ##########################--->
47
48  <!--## Identificaci n de objetos con haarcascade ##-->
49
50  <node name="tracker_node" pkg="stereo_color_tracker" type="
        disparity_track.py" output="screen">
51  <remap from="/stereo/left/image_raw" to="/stereo/left/
        image_rect_color"/>
52  <remap from="/stereo/right/image_raw" to="/stereo/right/
        image_rect_color"/>
53  </node>
54
55  <!--Hasta aqui-->
56
57
58  <!--################## Visualizaci n de im genes capturadas por
        las c maras usb haciendo uso de image_view ###############3-->
```

```
59
60
61
62    <node name="image_view_left" pkg="image_view" type="image_view"
        respawn="false" output="screen">
63      <remap from="image" to="/stereo_tracker/left"/>
64      <param name="autosize" value="true" />
65    </node>
66
67
68    <node name="image_view_right" pkg="image_view" type="image_view"
        respawn="false" output="screen">
69      <remap from="image" to="/stereo_tracker/right"/>
70      <param name="autosize" value="true" />
71    </node>
72  </launch>
```

Listing 3.4: Launch que realiza las tareas de detección de objetos y calcula su profundidad con modelo de segmentación de color

```
1
2  <!--################ Launch principal para estimaci n de
       profundidad ##################-->
3
4  <launch>
5
6    <!-- ######################## Inicializaci n de c maras USB
       haciendo uso de paqueter a usb_cam ###################-->
7
8  <group ns="stereo">
9    <node name="left" pkg="usb_cam" type="usb_cam_node" output="
       screen" >
10     <param name="video_device" value="/dev/video2" />
11     <param name="image_width" value="640" />
12     <param name="image_height" value="480" />
13     <param name="framerate" value="10" />
14     <param name="pixel_format" value="yuyv" />
15     <param name="camera_frame_id" value="left" />
16     <param name="io_method" value="mmap"/>
17      <param name="camera_info_url" value="file:///home/laura/
       Escritorio/CALIBRACIONMAYO/2505/left.yaml"/>
18   </node>
19
20   <node name="right" pkg="usb_cam" type="usb_cam_node" output="
       screen" >
21     <param name="video_device" value="/dev/video4" />
22     <param name="image_width" value="640" />
23     <param name="image_height" value="480" />
24     <param name="framerate" value="10" />
25     <param name="pixel_format" value="yuyv" />
26     <param name="camera_frame_id" value="right" />
27     <param name="io_method" value="mmap"/>
28     <param name="camera_info_url" value="file:///home/laura/
       Escritorio/CALIBRACIONMAYO/2505/right.yaml"/>
29   </node>
30
31
```

```
<!--###################### Rectificaci n de c maras
este nodo de stero image proc publica las imagenes rectificadas que
     se llaman image rect color ################################-->


<node name="stereo_image_proc" pkg="stereo_image_proc" type="
    stereo_image_proc" output="screen" >
<param name="queue_size" value="10" />
<param name="approximate_sync" value="True" />
<remap from="/stereo/left/image" to="/stereo/left/image_raw"/>
<remap from="/stereo/right/image" to="/stereo/right/image_raw"/>
</node>

</group>

<!-- AQui debemos incluir la instruccion de stereoproc paraobtener
    imagenes calbradas-->

<!--TOdo ESTO PERTENECE A LA RED NEURONAL-->
<!-- ## Identificaci n de objetos con Red Neuronal ## -->

<!-- # Nodos de detecci n con red neuronal #-->


<arg name="transport" default="compressed"/>


    <node pkg="dnn_detect" name="left_detect" type="dnn_detect"
    output="screen" respawn="false">
      <param name="image_transport" value="$(arg transport)"/>
      <param name="publish_images" value="true" />
      <param name="data_dir" value="$(find dnn_detect)/model"/>
      <remap from="/camera/compressed" to="/stereo/left/image_raw/$
    (arg transport)"/>
      <remap from="/camera_info" to="/stereo/left/camera_info"/>
    </node>


    <node pkg="dnn_detect" name="right_detect" type="dnn_detect1"
    output="screen" respawn="false">
      <param name="image_transport" value="$(arg transport)"/>
      <param name="publish_images" value="true" />
      <param name="data_dir" value="$(find dnn_detect)/model"/>
      <remap from="/camera/compressed" to="/stereo/right/image_raw/
    $(arg transport)"/>
      <remap from="/camera_info" to="/stereo/right/camera_info"/>
    </node>

<node name="tracker_node" pkg="stereo_color_tracker" type="
    nn_v1_track.py" output="screen">
     </node>

<!--Hata aqui-->


<!--################## Visualizaci n de im genes capturadas por
    las c maras usb haciendo uso de image_view ###############3-->
```

```
80
81
82
83    <node name="image_view_left" pkg="image_view" type="image_view"
        respawn="false" output="screen">
84      <remap from="image" to="/stereo_tracker/left"/>
85      <param name="autosize" value="true" />
86    </node>
87
88
89    <node name="image_view_right" pkg="image_view" type="image_view"
        respawn="false" output="screen">
90      <remap from="image" to="/stereo_tracker/right"/>
91      <param name="autosize" value="true" />
92    </node>
93
94  </launch>
```

Listing 3.5: Launch que realiza las tareas de detección de objetos y calcula su profundidad con Red Neuronal

Es importante señalar que la paquetería utilizada para la identificación de objetos por color puede hallarse en el siguiente repositorio de GitHub:

$$https://github.com/Elucidation/StereoColorTracking.$$

Procesamiento de imágenes

La implementación de algoritmos basados en áreas en OpenCV se realizan mediante los siguientes 3 pasos:

1. Prefiltrado
2. Búsqueda de correspondencia
3. Post-filtrado

Los parámetros controlan varios aspectos del algoritmo de coincidencia de bloques utilizado por stereo_image_proc para encontrar correspondencias entre las imágenes izquierda y derecha

La etapa de prefiltrado para normalizar el brillo de la imagen y mejorar la textura.

- texture_threshold : filtra las áreas que no tienen suficiente textura para una coincidencia confiable
- prefilter_size y prefilter_cap: controlan una fase de filtrado previo que normaliza el brillo de la imagen y mejora la textura en preparación para la coincidencia de bloques

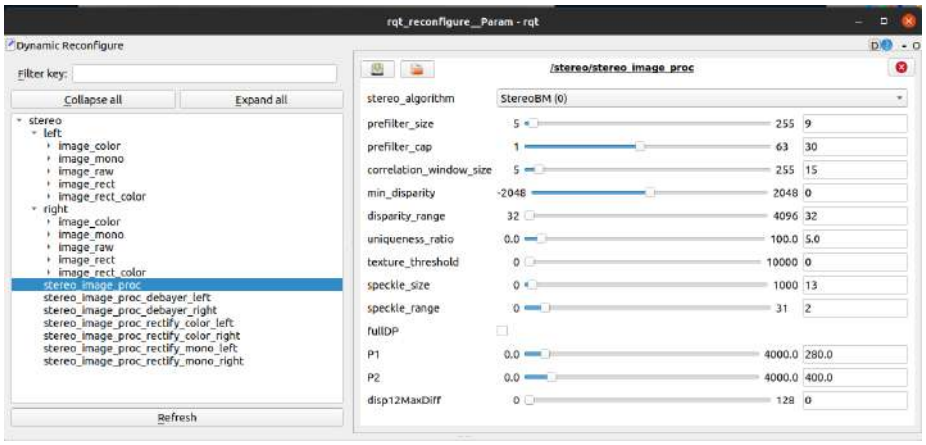


Figura 3.13: Párametros de procesamiento estereo

Si los parametros de fitrado se establecen demasiado altos, entonces lo que se observara es una imagen de disparidad vacia.

- iniqueness_ratio :controla un paso posterior al filtrado .

  Dado que la correspondencia se calcula por el método basado en área, por bloques en la que el rango de búsqueda esta definido por los parametros;

- disparity_range : es cuántos píxeles deslizar la ventana encima. Cuanto más grande es, mayor es el rango de profundidades visibles, pero se requiere más cálculo.

- min_disparity : controla el desplazamiento desde la pocisión x del píxel izquierdo, en el que se comieza la búsqueda.

- correlation_window_size : controla el tamaño de la ventana deslizante SAD (suma de diferencias absolutas) utilizada para encontrar puntos de coincidencia entre las imágenes izquierda y derecha. Los tamaños de ventana más grandes suavizarán los pequeños espacios en la imagen de disparidad, pero también mancharán los límites de los objetos

  Se utilizan para mejorar la etapa anterior, procesando la imagen de disparidad con un filtro de manchas.

- speckle_size : es el número de píxeles por debajo del cual una mancha de disparidad se descarta como moteado

- speckle_range : controla cuan cercanas deben ser las disparidades de un valor para ser consideradas parte del mismo bloque

  Resulta díficil analiticamente valores la precisión de la calibración porque todas las medidas tomadas por el sistema se visión dependen de los resultados de la calibració. Una de las formas en las que se puede valorar, es

de acuerdo al error enen la localizacion de puntos conocidos en el mundo 3D, una vez se han calibrado ambas cámaras y utilizando la triangulacion estereoscopica se miden la posiciones de puntos en la escena 3D. Dichas posiciones que son conocidas con cierta presición.

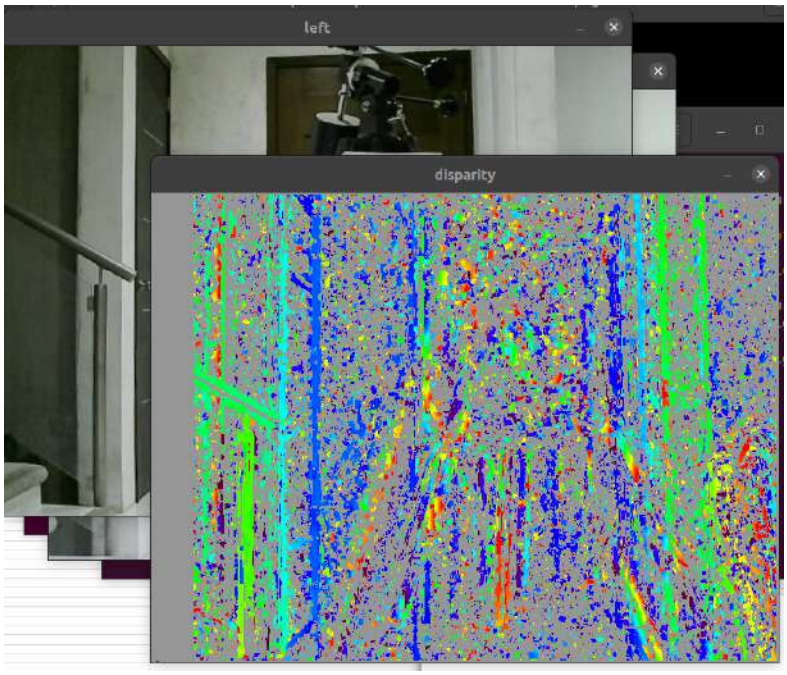


Figura 3.14: El mapa de disparidad generado por la diferencia de ubicación del mismo punto observado en las dos cámaras.

En la 3.14 si el color del pixel es mas claro, indica que el objeto esta mas cercano.

Así cuanto más cálido (cercano al rojo) es el color, mayor es el valor de su disparidad, mientras que cuanto más frio (cercano al azul) sea el color, menor será el valor de su disparidad. Una simple inspección visual de los mapas de disparidad en relación al par estereoscópico original permite deducir sin mucha dificultad la evidencia de que a mayor disparidad más proximidad de los objetos con respecto al sistema estereoscópico que captura las imágenes.

3.7. Grafos

Los siguientes grafos obtenidos con ayuda de ROS son un representación gráfica de como se comunican los nodos durante un proceso en ejecución. En

otras palabras, ilustran los nodos activos y los tópicos a los que se suscriben o que publican, dependiendo de la técnica de detección de objetos utilizada.

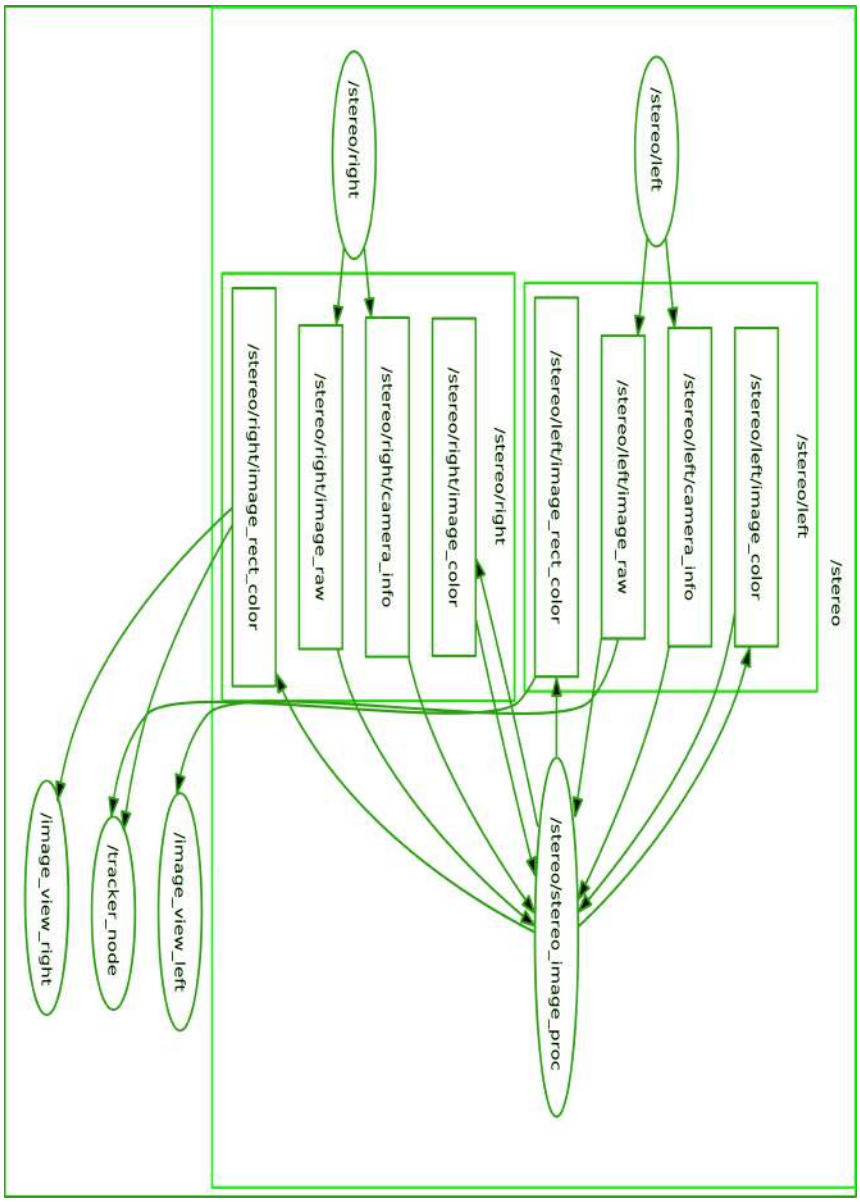


Figura 3.15: Comunicación entre nodos cuando se activan dos dispositivos de captura de imagen.

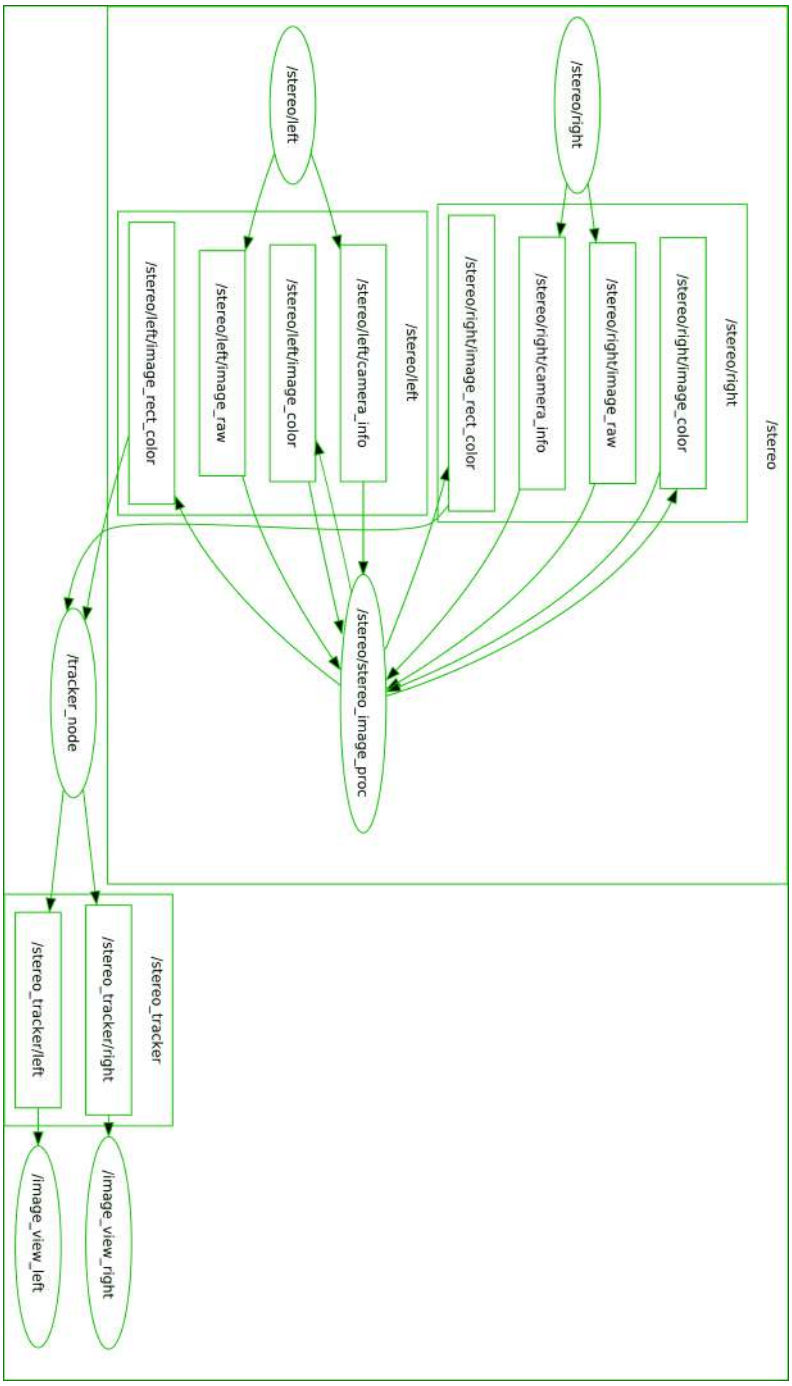


Figura 3.16: Comunicación entre nodos durante el proceso de detección de objetos con segmentación de color y posterior estimación de profundidad.

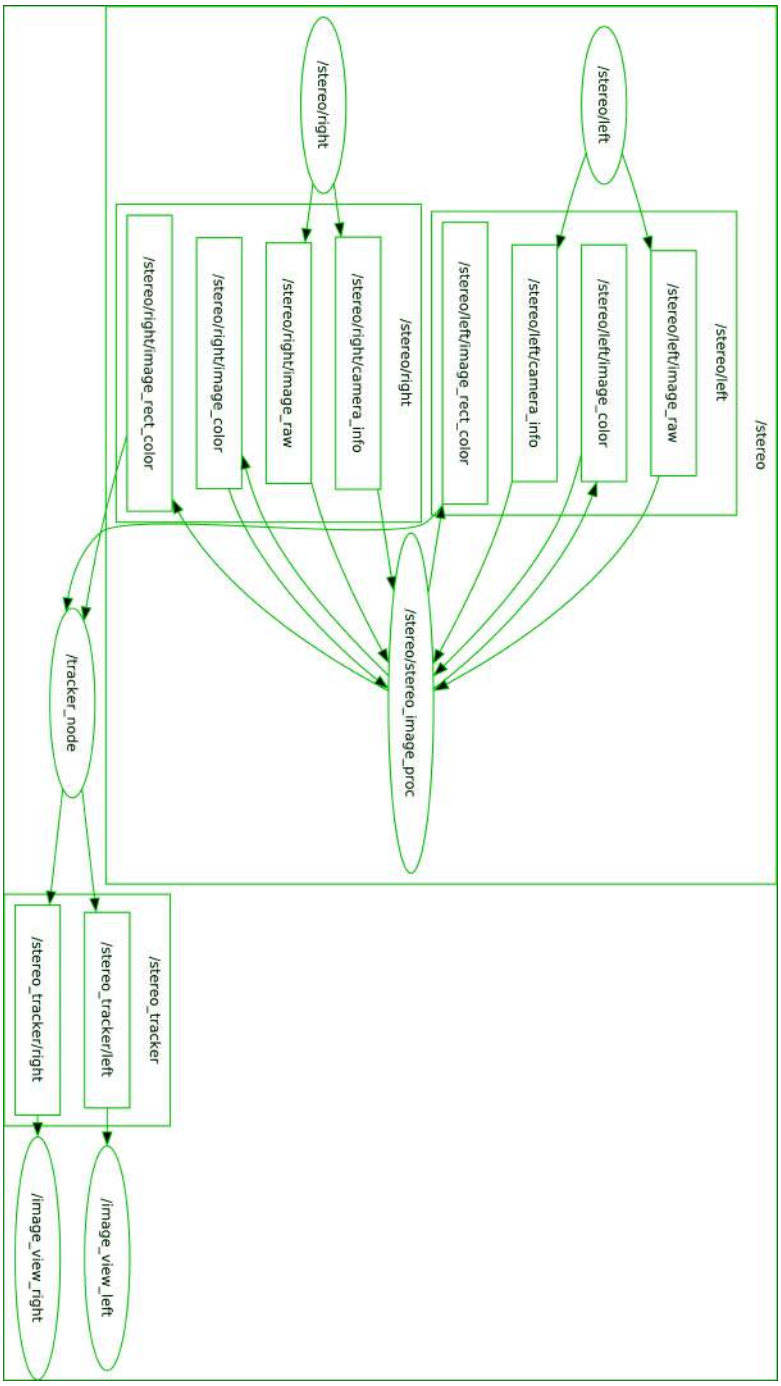


Figura 3.17: Comunicación entre nodos durante el proceso de detección de objetos con haarcascades y posterior estimación de profundidad.

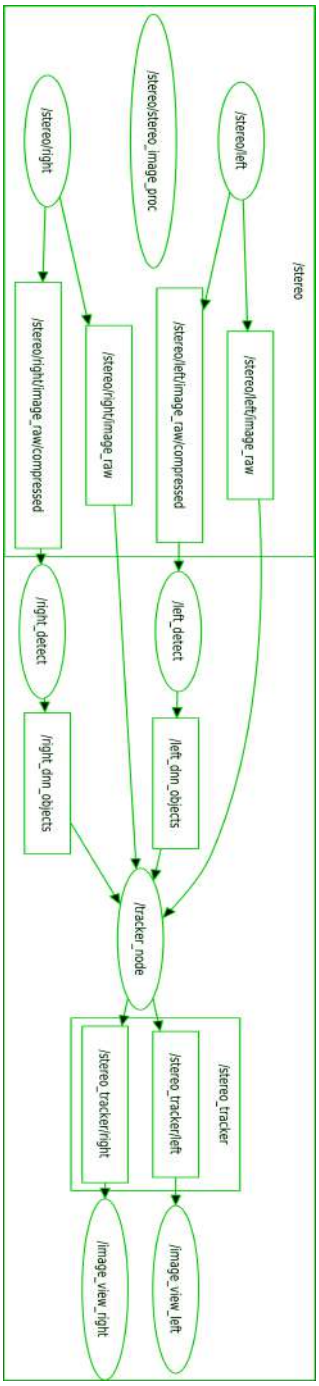


Figura 3.18: Comunicación entre nodos durante el proceso de detección de objetos con red neuronal convolucional y posterior estimación de profundidad.

Bibliografía

[1] K Schwab. The fourth industrial revolution: what it means, how to respond. https://www.weforum.org/agenda/2016/01/the-fourth-industrial-revolutionwhat-it-means-and-how-to-respond/, 2016.

[2] Dirk Helbing. Societal, economic, ethical and legal challenges of the digital revolution: from big data to deep learning, artificial intelligence, and manipulative technologies. In *Towards digital enlightenment*, pages 47–72. Springer, 2019.

[3] Anthony Elliott. *The culture of AI: Everyday life and the digital revolution*. Routledge, 2019.

[4] Assunta Di Vaio, Rosa Palladino, Rohail Hassan, and Octavio Escobar. Artificial intelligence and business models in the sustainable development goals perspective: A systematic literature review. *Journal of Business Research*, 121:283–314, 2020.

[5] Carlos Castedo Hernández, Rafael Estop Remacha, Lidia Santos, et al. Sistema de visión estereoscópico para el guiado de un robot quirúrgico en operaciones de cirugía laparosócopica hals. *Actas de las XXXVIII Jornadas de Automática*, 2017.

[6] Juan F Cortes Zarta, Yesica A Giraldo Tique, and Carlos F Vergara Ramírez. Red neuronal convolucional para la percepción espacial del robot inmoov a través de visión estereoscópica como tecnología de asistencia. *Enfoque UTE*, 12(4):88–104, 2021.

[7] Jairo Jener Pirca Cárdenas. Integración de un sistema robótico asistencial controlado mediante una interfaz cerebro computador para personas con discapacidad motora. *Pontificia Universidad Católica del Perú*, 2019.

[8] Luini Leonardo Hurtado-Cortes, John Alejandro Forero-Casallas, and Víctor Elberto Ruiz-Rosas. Artificial vision applied to manufacturing process/vision artificial aplicada al proceso de manufactura. *Visión electrónica*, 15(1):NA–NA, 2021.

[9] G Nick. Artificial intelligence statistics [updated for 2021]. https://techjury.net/blog/ai-statistics/gref, 2021.

[10] Elsa Adriana Cárdenas Quiroga, Luz Yolanda Morales Martín, and Andrés Ussa Caycedo. La estereoscopía, métodos y aplicaciones en diferentes áreas del conocimiento. *Revista Científica General José María Córdova*, 13(16):201–219, 2015.

[11] Santiago Martín, Javier Suárez, Ramón Rubio, and Ramón Gallego. Aplicación de los sistemas de visión estereoscópica en las enseñanzas técnicas. *Escuela de Ingenieros Técnicos Industriales de Gijón. Universidad de Oviedo*, 2004.

[12] Álvaro M Pons Moreno and Francisco M Martínez Verdú. *Fundamentos de visión binocular*, volume 74. Universitat de València, 2014.

[13] Enrique Alegre, Gonzalo Pajares, and Arturo de la Escalera. Conceptos y métodos en visión por computador. *España: Grupo de Visión del Comité Español de Automática (CEA)*, 2016.

[14] James Kobielus. Powering AI: The explosion of new AI hardware accelerators. https://www.infoworld.com/article/3290104/powering-ai-the-explosion-of-new-ai-hardware-accelerators.html, 2018. Accessed: 2022-02-08.

[15] Xin Feng, Youni Jiang, Xuejiao Yang, Ming Du, and Xin Li. Computer vision algorithms and hardware implementations: A survey. *Integration*, 69:309–320, 2019.

[16] Noor A Ibraheem, Mokhtar M Hasan, Rafiqul Z Khan, and Pramod K Mishra. Understanding color models: a review. *ARPN Journal of science and technology*, 2(3):265–275, 2012.

[17] Rodríguez Morales Roberto y Sosa Azuela Juan Humberto. *Procesamiento y anális digital de imágenes*. Alfa Omega Grupo Editor, S.A. de C.V, 2002.

[18] Intel Corporation. Cascade classifier. https://docs.opencv.org/3.4. Accessed: 2023-04-25.

[19] Pedro Ponce. *Inteligencia artificial: con aplicaciones a la ingeniería*. Alpha Editorial, 2010.

[20] David E Rumelhart, Richard Durbin, Richard Golden, and Yves Chauvin. Backpropagation: The basic theory. *Backpropagation: Theory, architectures and applications*, pages 1–34, 1995.

[21] Roberto Martín López et al. Detección de personas en imágenes de profundidad mediante redes neuronales convolucionales. 2019.

[22] Yang Liu, Jie Jiang, Jiahao Sun, Liang Bai, and Qi Wang. A survey of depth estimation based on computer vision. In *2020 IEEE Fifth International Conference on Data Science in Cyberspace (DSC)*, pages 135–141. IEEE, 2020.

[23] Nasir Saeed, Shuaishuai Guo, Ki-Hong Park, Tareq Y Al-Naffouri, and Mohamed-Slim Alouini. Optical camera communications: Survey, use cases, challenges, and future trends. *Physical Communication*, 37:100900, 2019.

[24] Sameer Ansari, Neal Wadhwa, Rahul Garg, and Jiawen Chen. Wireless software synchronization of multiple distributed cameras. In *2019 IEEE International Conference on Computational Photography (ICCP)*, pages 1–9. IEEE, 2019.

[25] Juan Pablo Flores-Flores and Rafael Martínez-Guerra. Dynamical distributed control and synchronization. *Nonlinear Dynamics*, 103(2):1663–1679, 2021.

[26] Rafael Martínez-Guerra and Juan Pablo Flores-Flores. *An Approach to Multi-agent Systems as a Generalized Multi-synchronization Problem.* Springer Nature, 2023.

[27] Pajares Martínsanz Gonzalo y De la Cruz García Jeús M. *Visión por computador: imágenes digitales y aplicaciones.* Alfa Omega Grupo Editor, S.A. de C.V, 2002.

[28] Pablo Vera and Joaquín Salas. Determinacion de la incertidumbre de los parametros intrinsecos del modelo de una camara. *Centro de Investigación en Ciencia Aplicada y Tecnología Avanzada (CICATA) del IPN, Simposio de Metrología*, 2004.

[29] Andrea Fusiello, Emanuele Trucco, and Alessandro Verri. A compact algorithm for rectification of stereo pairs. *Machine vision and applications*, 12:16–22, 2000.

[30] Richard Szeliski. *Computer vision: algorithms and applications.* Springer Science & Business Media, 2010.

[31] Diego Alfonso Yhama Valero. La imagen flotante:"principios sobre estereoscopía". *Pontificia Universidad Javeriana*, 2010.

[32] Standford's Artificial Intelligence Laboratory Willow Garage. Robot operating system. https://www.ros.org/. Accessed: 2023-07-20.

[33] Microsoft. Github. https://github.com/. Accessed: 2023-07-20.

[34] Domínguez Míngues Tomás. *Visión Artificial.* Alfa Omega Grupo Editor, S.A. de C.V, 2022.

[35] Timothy A Clarke and John G Fryer. The development of camera calibration methods and models. *The Photogrammetric Record*, 16(91):51–66, 1998.

[36] Ross Girshick Ali Harhali Joseph Redmon, Santosh Divvala. "you only look once: Unified, real-time object detection.". https://arxiv.org/abs/1506.02640. Accessed: 28-09-25.

[37] J Gallego, Patricia Compañ, Pilar Arques, Carlos Villagrá, and Rafael Molina. Detección de objetos y estimación de su profundidad mediante un algoritmo de estéreo basado en segmentación. In *WAF (VIII Workshop de Agentes Físicos)-CEDI*, 2007.

[38] Rafael Solar Hernandez. Transferencia de aprendizaje con vgg16 para la evaluación del algoritmo de explicación de ia lime. *Benemétita Universidad Autónoma de Puebla*.

[39] Ana Luisa Ballinas. Algoritmode visión artificial para detección de reductores de velocidad en superficies viales. 2022.