Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería Mecánica Eléctrica

DISEÑO E IMPLEMENTACIÓN DE UN PROTOTIPO DE VEHÍCULO AUTÓNOMO DESARROLLADO A PARTIR DE ALGORITMO DE VISIÓN COMPUTARIZADA REDES NEURONALES Y SENSORES

José Fernando Pérez Pérez
Asesorado por el Ing. Byron Odilio Arrivillaga Méndez

Guatemala, julio de 2021

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

DISEÑO E IMPLEMENTACIÓN DE UN PROTOTIPO DE VEHÍCULO AUTÓNOMO DESARROLLADO A PARTIR DE ALGORITMO DE VISIÓN COMPUTARIZADA REDES NEURONALES Y SENSORES

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

JOSÉ FERNANDO PÉREZ PÉREZ
ASESORADO POR EL ING. BYRON ODILIO ARRIVILLAGA MÉNDEZ

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO ELECTRÓNICO

GUATEMALA, JULIO DE 2021

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANA	Inga. Aurelia Anabela Cordova Estrada
VOCAL I	Ing. José Francisco Gómez Rivera
VOCAL II	Ing. Mario Renato Escobedo Martínez
VOCAL III	Ing. José Milton de León Bran
VOCAL IV	Br. Christian Moisés de la Cruz Leal
VOCAL V	Br. Kevin Armando Cruz Lorente
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Pedro Antonio Aguilar Polanco
EXAMINADOR	Ing. Byron Odilio Arrivillaga Méndez
EXAMINADOR	Ing. José Aníbal Silva de los Ángeles
EXAMINADOR	Ing. Gustavo Benigno Orozco Godínez
SECRETARIA	Inga. Lesbia Magalí Herrera López

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

DISEÑO E IMPLEMENTACIÓN DE UN PROTOTIPO DE VEHÍCULO AUTÓNOMO DESARROLLADO A PARTIR DE ALGORITMO DE VISIÓN COMPUTARIZADA REDES NEURONALES Y SENSORES

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería Mecánica Eléctrica, con fecha 24 de octubre de 2019.

José Fernando Pérez Pérez

Guatemala 27 de noviembre 2020

Ingeniero Julio César Solares Peñate
Coordinador del Área de Electrónica
Escuela de Ingeniería Mecánica Eléctrica
Universidad de San Carlos de Guatemala

Me permito dar aprobación al trabajo de graduación titulado **DISEÑO E IMPLEMENTACIÓN DE UN PROTOTIPO DE VEHÍCULO AUTÓNOMO DESARROLLADO A PARTIR DE ALGORITMO DE VISIÓN COMPUTARIZADA REDES NEURONALES Y SENSORES,** elaborado por el estudiante José Fernando Pérez Pérez quien se identifica con el número de DPI 2821 64871 0101 y carnet 201404272, por considerar que cumple con los requisitos establecidos.

Por tanto, el autor de este trabajo y, yo, como su asesor, nos hacemos responsables por el contenido y conclusiones del mismo.

Atentamente,

Ing. Byron Odilio Arrivillaga Méndez
Colegiado activo No. 5217
Asesor

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

Guatemala, 30 de noviembre de 2020

Señor Director
Armando Alonso Rivera Carrillo
Escuela de Ingeniería Mecánica Eléctrica
Facultad de Ingeniería, USAC

Estimado Señor Director:

Por este medio me permito dar aprobación al Trabajo de Graduación titulado **DISEÑO E IMPLEMENTACIÓN DE UN PROTOTIPO DE VEHÍCULO AUTÓNOMO DESARROLLADO A PARTIR DE ALGORITMO DE VISIÓN COMPUTARIZADA REDES NEURONALES Y SENSORES,** desarrollado por el estudiante **José Fernando Pérez Pérez**, ya que considero que cumple con los requisitos establecidos.

Sin otro particular, aprovecho la oportunidad para saludarlo.

Atentamente,

ID Y ENSEÑAD A TODOS

Ing. Julio César Solares Peñate
Coordinador de Electrónica

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA MECÁNICA ELÉCTRICA

REF. EIME 78. 2021.

El Director de la Escuela de Ingeniería Mecánica Eléctrica, después de conocer el dictamen del Asesor, con el Visto Bueno del Coordinador de Área, al trabajo de Graduación del estudiante; JOSÉ FERNANDO PÉREZ PÉREZ titulado; DISEÑO E IMPLEMENTACIÓN DE UN PROTOTIPO DE VEHÍCULO AUTÓNOMO DESARROLLADO A PARTIR DE ALGORITMO DE VISIÓN COMPUTARIZADA REDES NEURONALES Y SENSORES, procede a la autorización del mismo.

Ing. Armando Alonso Rivera Carrillo

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION ESCUELA DE INGENIERIA MECANICA ELECTRICA
FACULTAD DE INGENIERIA

GUATEMALA, 20 DE ABRIL 2,021.

Decanato
Facultad de Ingeniería
24189101 - 24189102

DTG. 319-2021

La Decana de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería Mecánica Eléctrica, al Trabajo de Graduación titulado: **DISEÑO E IMPLEMENTACIÓN DE UN PROTOTIPO DE VEHÍCULO AUTÓNOMO DESARROLLADO A PARTIR DE ALGORITMO DE VISIÓN COMPUTARIZADA REDES NEURONALES Y SENSORES**, presentado por el estudiante universitario**: José Fernando Pérez Pérez,** y después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, autoriza la impresión del mismo.

IMPRÍMASE:

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANA
FACULTAD DE INGENIERÍA

Inga. Anabela Cordova Estrada
Decana

Guatemala, julio de 2021

AACE/cc

ACTO QUE DEDICO A:

Dios	Por permitirme llegar a este punto en mi vida, otorgarme la sabiduría y fuerza para alcanzar esta meta.
Mis padres	Melva Esperanza Pérez Mejía y José Alfredo Pérez Melgar (q. e. p. d). Por ser mi más grande inspiración y por su apoyo incondicional.
Mis hermanas	Karla Esperanza y Laura Mariella Pérez por apoyarme, brindarme su cariño y comprensión en cada etapa de este camino.
Mis abuelos	Candida Eloisa Toledo y Martín Alfredo Pérez (q. e. p. d); María del Carmen Mejía y Lauro Pérez de León, por compartir conmigo sus experiencias y brindarme su consejo cuando lo necesite.
Mis tíos y primos	Por su apoyo y consejo.
Mi novia	Rosa Andrea Cúmez Caté, por estar presente durante mi carrera y brindarme su apoyo incondicionalmente.

AGRADECIMIENTOS A:

Universidad de San Carlos de Guatemala	Por ser la casa de estudios que me albergo durante mi carrera y me proporciono las herramientas y conocimientos para culminar esta etapa.
Facultad de Ingeniería	Por ser la fuente de conocimientos que permitieron mi formación académica.
Mi asesor	Ing. Byron Arrivillaga, por compartir sus conocimientos conmigo y brindarme su tiempo para la realización del presente trabajo.
Mis amigos	Por hacer mi vida estudiantil amena y compartir esta etapa conmigo.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

LISTA DE SÍMBOLOS

Símbolo	Significado
A	Amperio
AVI	*Audio Video Interleave*, formato de audio y video propio de *Microsoft.*
BMP	*Bit map* o mapa de bits.
CV	*Computer vision* o visión por computadora.
FLV	*Flash Video*, formato de video utilizado para transmitir video por Internet usando el complemento *Adobe.*
FPS	*Frames per Second* o cuadros por segundo.
GND	*Ground,* tierra o punto de referencia eléctrico.
H5	Formato de archivos para almacenar estructura de datos numéricos utilizado en *Python.*
Hz	Hercio, unidad de frecuencia.
HSV	*Hue, Sauturation and Value*, modelo de color basado en los componentes de tinte, matiz y saturación.
JPEG	*Joint Photographic Experts Group*
kHz	Kilohertz
PNG	*Portable Network Graphics*
TIFF	*Tagged image file format*
Trig	*Trigger* o señal de activación.
Vcc	Voltaje aplicado al colector del transistor.
V	Voltio
XML	*Extensible Markup Language,* lenguaje de marcado utilizado para codificación de documentos.

GLOSARIO

Algoritmo Conjunto de instrucciones ordenadas secuencialmente para realizar una tarea específica.

BSD *Berkeley Software Distribution*, sistema operativo derivado de *UNIX.*

Gradiente Generalización del concepto de derivada matemática para funciones de varias variables. Indica la dirección de máximo cambio para una función dada.

Kernel Matriz de convolución utilizada para aplicación de distintos tipos de filtros a imágenes.

Pixel Unidad más pequeña de una imagen digital.

Puente H Circuito eléctrico utilizado generalmente para permitir a un motor girar en ambos sentidos.

Raspberry Pi Ordenador de placa reducida de bajo coste.

RESUMEN

En este trabajo de graduación se presenta la serie de pasos necesarios para crear un prototipo de vehículo a escala que pueda controlarse de forma autónoma a partir de la digitalización de imágenes en su entorno.

Se enseñarán las técnicas básicas de procesamiento de imágenes digitales de forma teórica y práctica utilizando la librería *OpenCV,* también se explican los conceptos básicos para dar una breve introducción a las redes neuronales artificiales y sus casos de uso.

El vehículo a escala cuenta con una interfaz de *software* que permite controlar una cámara para obtener las imágenes en tiempo real y 2 motores, uno para controlar la dirección y otro que controla la aceleración del vehículo.

Utilizando la cámara se obtuvo un conjunto de imágenes y se aplicaron técnicas de procesamiento de imágenes digitales. A cada imagen procesada se le agrego una etiqueta que representa la dirección de movimiento en el momento de capturar la imagen, con el objetivo de crear un conjunto de datos adecuado para entrenar un modelo computacional, utilizando redes neuronales artificiales, que permite clasificar la imagen y predecir la dirección del movimiento.

Se implemento una pista modelo y se obtuvo un conjunto de 547 imágenes con su etiqueta respectiva; se entrenó un modelo computacional con una exactitud en la predicción del 62,7 % de los datos utilizados para prueba, 110 imágenes del conjunto original, lo que significa que 5 de cada 8 imágenes son clasificadas de manera correcta.

OBJETIVOS

General

Diseñar un prototipo de vehículo a escala que sea capaz de interpretar las características de su entorno para tomar decisiones que permitan la conducción sin intervención humana.

Específicos

1. Presentar los fundamentos necesarios para el procesamiento de imágenes utilizando una biblioteca libre de visión artificial *OpenCV.*

2. Presentar una breve introducción a las redes neuronales.

3. Crear un ambiente de *software/hardware* en un ordenador de placa reducida con puertos de entrada y salida, que pueda obtener imágenes mediante una cámara y mediciones de sensores, para enviar señales digitales que controlen el movimiento de un vehículo a escala.

4. Crear un modelo computacional entrenado a partir de imágenes del entorno para controlar el vehículo a escala.

INTRODUCCIÓN

Desde la época de los vehículos propulsados a vapor, hasta los autos eléctricos más complejos diseñados en la actualidad; el ser humano ha logrado modificar los vehículos de modo que se ajusten a sus necesidades y gustos. El principal objetivo de los automóviles, es ser el medio de transporte cotidiano de los seres humanos. A medida que la tecnología va siendo cada vez más sofisticada se han hecho avances notables en el área automotriz, desde hacer más eficiente el sistema de combustión y las medidas de seguridad mediante componentes electrónicos, hasta la creación de vehículos capaces de movilizarse de un punto A hacia un punto B sin un conductor a bordo, calificados como vehículos autónomos. Varios grupos de ingeniería desconocidos trabajaron en este tipo de vehículos desde 1980, y en la actualidad ya existen varios fabricantes que destacan por sistemas autónomos en sus vehículos.

El desarrollo de este trabajo de investigación consiste en diseñar un prototipo de vehículo a escala que pueda tomar sus propias decisiones para movilizarse de un punto a otro, sin intervención humana, definiendo como parámetros de entrada las imágenes del recorrido y sensores de distancia; estos datos serán procesados utilizando algoritmos de visión computarizada y redes neuronales, para entrenar un modelo que sea capaz de decirle al dispositivo qué decisión tomar con base al entorno en el que se encuentra. Asimismo, este trabajo de investigación tiene como objetivo estimular e incentivar al estudiante a sumergirse en los temas de visión por computadora, y los nuevos avances que se tienen con la inteligencia artificial en la búsqueda de crear dispositivos físicos o de software que puedan tomar decisiones por sí solos.

1. VISIÓN POR COMPUTADORA

1.1. Concepto

Conocida también como visión artificial, es una disciplina de la informática que incluye métodos para extraer y analizar la información de una imagen, un vídeo o cualquier otro tipo de dato multidimensional con el objetivo de generar información simbólica o una representación numérica que un ordenador sea capaz de entender para describir el mundo real o tomar una decisión.

Desde el punto de vista de ingeniería es tratar de automatizar las tareas que el sistema visual humano puede realizar; desde el punto de vista científico, su fin es ocuparse de la teoría detrás de los sistemas digitales que extraen la información de las imágenes, por ejemplo, optimizar los algoritmos utilizados para el procesamiento de imágenes.

Su campo de aplicación es cada vez más grande y variado, pasando por el reconocimiento de patrones en una imagen, segmentación de imágenes, reconocimiento facial como método de seguridad utilizado para desbloquear nuestro teléfono, guiar robots en distintos ambientes o la conducción automática de vehículos.

1.2. Software libre para visión por computadora

Para el desarrollo de este trabajo se presentan librerías de software utilizadas para visión por computador de uso libre, comúnmente llamadas open

source o código abierto que permiten al usuario final utilizar y/o modificar el código fuente.

OpenCV, por sus siglas en inglés *Open Source Computer Vision* es una biblioteca informática de código abierto que se utiliza para el procesamiento de imágenes o visión artificial; originalmente fue desarrollada por Intel en la década de los 90's. Se desarrolló en lenguaje C++ permitiendo que sea fácil de utilizar y altamente eficiente. Tiene interfaces que permiten su uso en C++, C, Python, Java y Matlab. Es multiplataforma y ofrece más de 500 funciones para el procesamiento de imágenes. Se utiliza comúnmente en sistemas de vigilancia, robótica y para investigación en el área de visión artificial.

Point Cloud Library, es una biblioteca informática de código abierto que se utiliza para el procesamiento de imágenes en la categoría de nubes de puntos. Contiene numerosos algoritmos que permiten aplicar filtros, estimación de características, reconstrucción de superficies y segmentación aplicado principalmente a nubes de puntos. Es un conjunto de librerías desarrolladas en C++, es multiplataforma.

Sus puntos a favor radican en que tienen una amplia comunidad de usuarios y desarrolladores activos, permitiendo resolver problemas de funcionamiento en períodos de tiempo cortos. Además, tienen su documentación actualizada y disponible para cualquier usuario incluyendo también bastantes tutoriales para incentivar su uso. La principal desventaja de estas librerías es que requieren conocimientos de programación y visión artificial bastante amplios para sacarle el máximo provecho.

Las librerías presentadas anteriormente están publicadas bajo una licencia de código abierto BSD, lo que permite que sean utilizadas libremente para fines

comerciales o de investigación siempre y cuando se cumplan las condiciones de este tipo de licencia.

1.3. Redes neuronales artificiales

Las redes neuronales artificiales son un modelo computacional basado en el conocimiento actual del funcionamiento de las redes neuronales biológicas principalmente de las neuronas y sus formas de transmitir la información en la red. Dicho modelo es capaz de aproximar una función que se ajuste a un conjunto de datos que sea difícil de modelar con matemáticas convencionales.

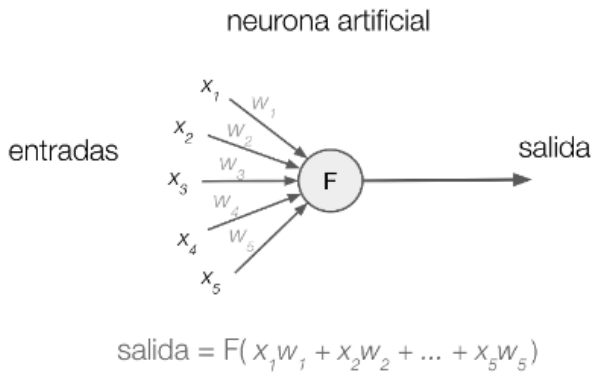
1.3.1. Neurona artificial

Es la unidad básica de una red neuronal, no es más que una función matemática; internamente la neurona artificial realiza una suma ponderada de todos los valores de entrada. La ponderación de cada entrada viene dada por el peso que se le asigna a cada una de las conexiones de entrada formando un vector de pesos que es el equivalente a las conexiones sinápticas de una neurona real. La actividad de la neurona consiste en generar una única salida aplicando una función de activación, F, a la salida de la suma ponderada obteniéndose la siguiente expresión:

$$\text{Salida} = F(w_1x_1+w_2x_2+\ldots+w_nx_n)$$

En la figura 1 se observa la representación gráfica de una neurona artificial, se puede observar que es una función que realiza una suma ponderada de sus entradas y aplica una función llamada activación al resultado de esta suma y eso se propaga hacia la salida de la neurona.

Figura 1. **Neurona artificial**



Fuente: PÉREZ, Jorge. *Neurona artificial con 5 entradas*. https://users.dcc.uchile.cl/~jperez/difusion/turing-award-2018.html#id.u7p2373ysdj3. Consulta: febrero 2020.

1.3.2. Funcionamiento

Las redes neuronales artificiales aprenden de sí mismas a partir de un conjunto de datos de entrada, en vez de ser programadas de forma explícita, sobresaliendo en áreas donde es difícil trabajar con programación convencional. Por ejemplo, clasificación de imágenes y texto.

Para que la red aprenda automáticamente se declara una función de pérdida que evalúa la red directamente, es decir, se inicializa la red con valores aleatorios y se evalúan los valores de entrada en la red obteniendo el valor de salida que se compara con el valor real o etiqueta de la entrada para cuantificar el valor de perdida. Obtenido este valor la red ejecutará una serie de algoritmos que modificarán el valor de los pesos en cada una de las capas para hacer que el valor de perdida sea muy cercano a cero.

1.3.3. Estructura

La estructura de la red nos dice la cantidad de neuronas artificiales que debe tener cada una de sus capas.

1.3.3.1. Capa de entrada

En esta capa se define el número de entradas que tendrá la red neuronal artificial. Por ejemplo, se quiere crear una red que nos ayudara a clasificar tipos de flores dados las siguientes características: largo, ancho, área y color del pétalo. Nuestra red debe tener una unidad por cada atributo, para este ejemplo deberíamos tener una capa con 4 unidades de entrada. Cada unidad contiene la información de un atributo distinto en forma numérica.

1.3.3.2. Capa de salida

La cantidad de neuronas artificiales en la capa de salida está dada por el número de clases que se identifiquen. Continuando con el ejemplo anterior, se asignará arbitrariamente una capa de salida con 2 neuronas, porque se sabe que trataremos con las siguientes flores: rosas y tulipanes. Si se proporcionan datos de una flor que no se encuentre en estas clases, la red retornará la clase con mayor probabilidad de parecerse a esta flor no clasificada.

1.3.3.3. Capas ocultas

Las capas ocultas contienen neuronas artificiales; existen varias reglas generales para determinar la cantidad de neuronas contenidas en la capa oculta, pero no existe una regla especifica.

Se procura que el número de neuronas artificiales en esta capa se encuentre en el intervalo comprendido por el tamaño de la capa de entrada y la de salida. Se puede tener más de una capa oculta. El método empírico es válido en estas circunstancias, la red se prueba con distintas arquitecturas y se elige el que mejor se adapte a nuestro problema.

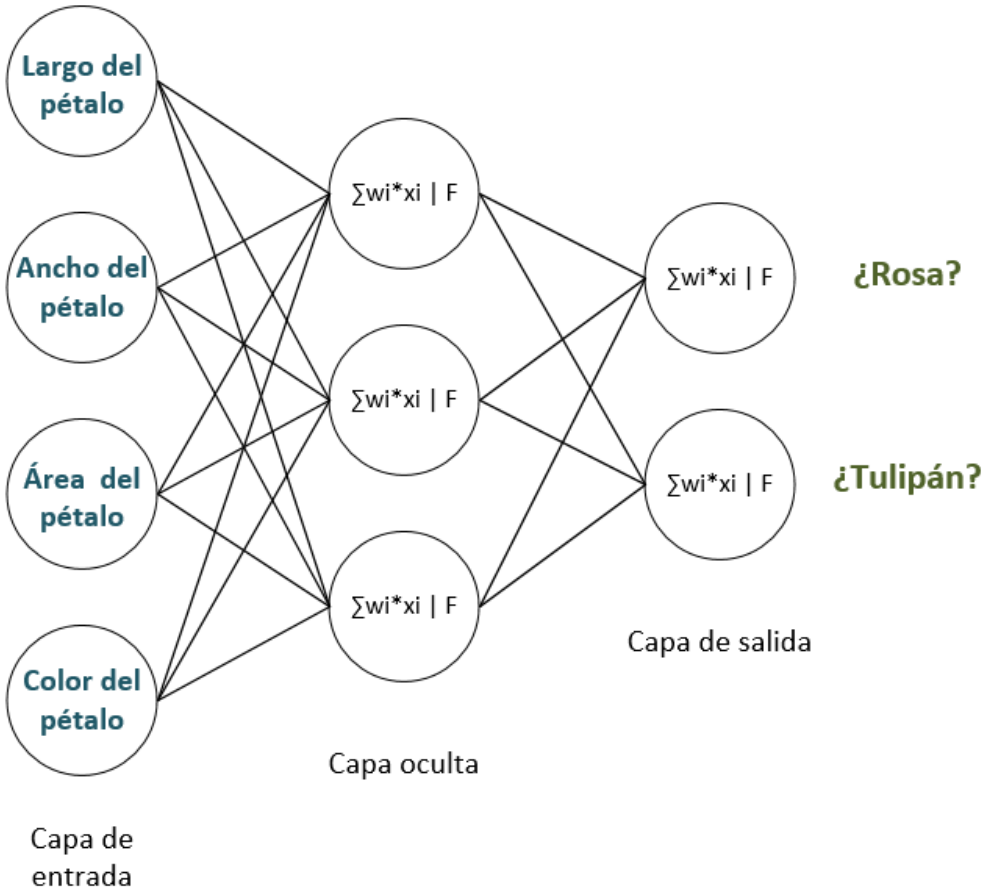
Un factor importante a tener en cuenta es el sobreajuste; esto ocurre cuando hay una cantidad excesiva de información contenida en las capas ocultas de la red, provocando que la misma se ajuste de una manera exacta a la información utilizada para entrenar la red haciendo que la aplicación del modelo con información desconocida no sea fiable.

Figura 2. **Sobreajuste**

Fuente: IBM Cloud Education. *Sobreajuste*. https://www.ibm.com/cloud/learn/overfitting. Consulta: 15 de febrero de 2020.

Como se observa en el modelo representado por la línea color roja de la figura 2 el mismo se ajusta de manera exacta al conjunto de datos proporcionados y esto puede tener como efecto que el modelo no pueda generalizar de forma correcta con datos no conocidos. Para nuestra red se quiere que el modelo sea más parecido al representado por la línea negra de esta figura.

Figura 3. **Red neuronal para clasificar flores**



Fuente: elaboración propia, empleando Microsoft Visio.

En la figura 3 se observa la red neuronal utilizada como ejemplo para explicar su estructura, la cual está diseñada con la siguiente arquitectura:

- Capa de entrada: 4 unidades, Correspondientes a las características de las flores.
- Capa de salida: 2 neuronas artificiales, correspondientes a las clases de clasificación planteadas.
- Capas ocultas: 1 capa con 3 neuronas artificiales.

1.3.4. Algoritmos de entrenamiento

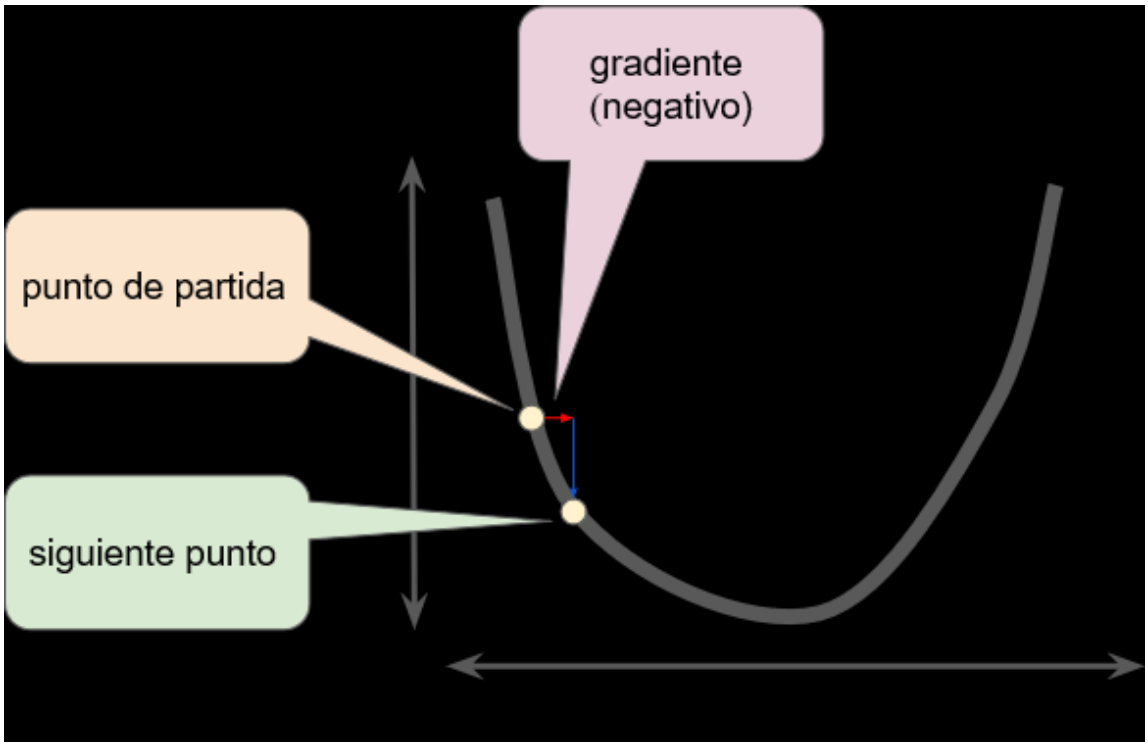
Los algoritmos de entrenamiento permiten a la red neuronal modificar sus parámetros en respuesta a la información de entrada, como se dijo antes, el objetivo de la red neuronal es minimizar la función de costo para tener una mejor aproximación a la función que compone los datos de entrenamiento. Matemáticamente se puede encontrar un mínimo de cualquier función utilizando la operación derivada de esa función, esto resulta relativamente sencillo cuando conocemos la función. Sin embargo, cuando se trata de redes neuronales se manejan funciones que dependen de miles y a veces millones de variables (los pesos de las capas) y es aquí cuando debemos recurrir a métodos numéricos para resolver el problema.

A continuación, se presentan 2 algoritmos que forman parte fundamental del entrenamiento de una red neuronal artificial:

- Descenso del gradiente: es un algoritmo de optimización que se utiliza en las redes neuronales para encontrar un mínimo de la función de coste. En matemáticas el vector gradiente de una función evaluado en cualquier punto genérico del dominio indica la dirección en la cual varía más

rápidamente. El gradiente se representa con el operador diferencial nabla. Para encontrar el mínimo de la función utilizando este algoritmo se calcula la derivada parcial, vector gradiente, respecto a cada parámetro en cada iteración, este valor nos indica el sentido en que se encuentra el mínimo más próximo. El valor de la derivada se le resta a cada uno de los parámetros multiplicado por una tasa de aprendizaje.

Figura 4. **Descenso del gradiente**

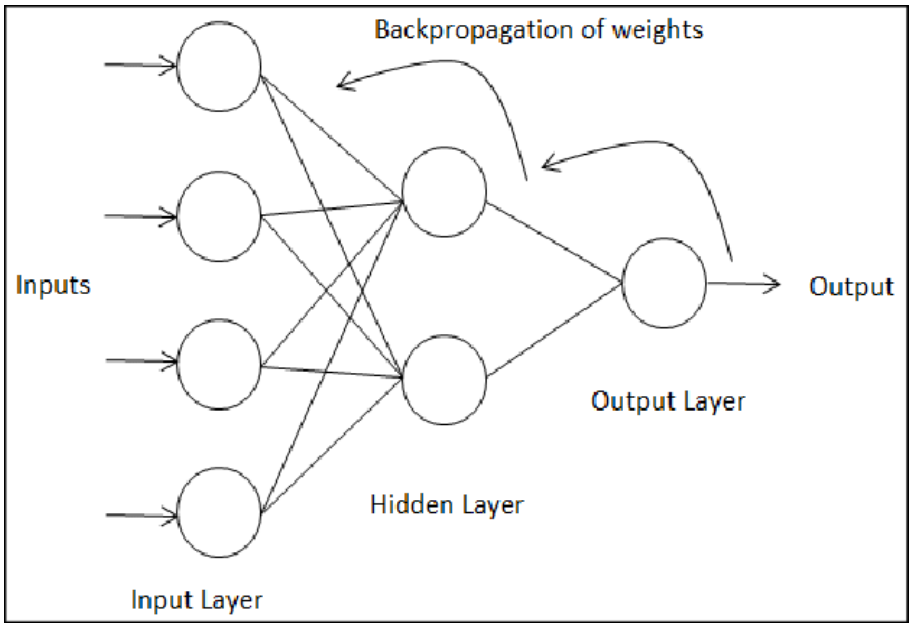


Fuente: Google Developers*. Descenso del gradiente*. https://developers.google.com/machine-learning/crash-course/reducing-loss/gradient-descent?hl=es. Consulta: 15 de febrero de 2020.

El descenso del gradiente se repite hasta que encuentre un valor que minimice la función de costo, puede ser en un mínimo local o uno global en el mejor de los casos.

- Propagación inversa**:** para calcular el descenso del gradiente se necesita conocer el vector gradiente de la función de coste que se define como la derivada parcial del coste con respecto a cada uno de los parámetros de la red. Por este motivo se creó el algoritmo de propagación inversa, el cual nos permite encontrar el vector gradiente que contiene los valores de estas derivadas. El algoritmo consiste en encontrar las derivadas parciales a partir de la función de coste en dirección contraria, es decir, primero se calculan estas derivadas con respecto a la capa anterior y así con todas las capas. Como es de esperar el error en la capa **n** depende del error en la capa posterior **n+1**. Entonces se calcula el error en cada neurona artificial asumiendo que el error total proviene de la capa posterior.

Figura 5. **Propagación inversa**



Fuente: GHAZALI, Rozaida. *Backpropagation*. https://www.researchgate.net/figure/The-structure-of-single-hidden-layer-MLP-with-Backpropagation-algorithm_fig2_234005707. Consulta: 16 de febrero de 2020

1.4. Introducción a *OpenCV*

OpenCV contiene una serie de módulos enfocados principalmente a la visión por computadora. A manera introductoria se enseñará como se utiliza esta librería para manejar archivos de imagen y video utilizando el lenguaje de programación Python en su versión 3.6.5.

1.4.1. Utilización de imágenes y cámaras

La mayoría de las aplicaciones en el ámbito de visión por computadora necesitan una imagen o video de entrada y muchas veces el resultado es una imagen o video de salida. Si la aplicación es interactiva es probable que se utilice una cámara digital como fuente de información de entrada. No necesariamente el resultado de nuestra aplicación de visón por computadora será una imagen o un video, también puede ser la activación de un sensor si nos referimos a electrónico o robótica, una acción con una base de datos o ejecutar una función a nivel de software.

1.4.1.1. Lectura y escritura de archivos de imagen

Las funciones *imread()* e *imwrite()* contenidas en la librería nos permiten la lectura y escritura de archivos de imagen, normalmente soportan los siguientes formatos: BMP, PNG, JPEG y TIFF.

En la figura 6 se muestra un ejemplo del código sobre cómo leer una imagen y guardar la misma imagen con un formato distinto que sea soportado.

Figura 6. **Ejemplo lectura y escritura de imágenes**

```python
import cv2 as cv

#Lectura de imagen
img = cv.imread('input/nasa.png')
#Escritura de imagen con distinto formato
img_converted = cv.imwrite('output/nasa.jpg')
```

Fuente: elaboración propia, empleando la herramienta https://carbon.now.sh.

1.4.1.2. Lectura y escritura de archivos de video

Las clases *VideoCapture* y *VideoWriter* contenidas en la librería nos permiten el manejo de archivos de video. Los formatos soportados varían dependiendo del sistema, pero siempre se incluye soporte para el formato AVI.

Mediante el método *read()* un objeto de la clase *VideoCapture* acumular en un *buffer* una nueva imagen hasta que el video llegue al final. De forma contraria, una imagen se puede enviar por el método *write()* de la clase VideoWriter el cual adjunta esta imagen a un archivo de video.

En la figura 7 se muestra un ejemplo de cómo se lee un archivo de video codificado en el formato AVI y ese mismo video se utilizará para crear uno nuevo con formato FLV.

Figura 7. **Ejemplo lectura y escritura de videos**

```python
import cv2 as cv

#Lectura de un archivo de video
videoLectura = cv.VideoCapture('input/video1.avi')
fps = videoLectura.get(cv.CAP_PROP_FPS)
size = (int(videoLectura.get(cv.CAP_PROP_FRAME_WIDTH)),
        int(videoLectura.get(cv.CAP_PROP_FRAME_HEIGHT)))

#Escritura de un archivo de video
videoEscritura = cv.VideoWriter(
    'output/video1.flv', cv.VideoWriter_fourcc('F','L','V','1'), fps, size)

flag, frame = videoLectura.read()
while flag:
    videoEscritura.write(frame)
    flag, frame = videoLectura.read()
```

Fuente: elaboración propia, empleando la herramienta https://carbon.now.sh.

1.4.1.3. Lectura de imágenes desde una cámara

En muchas aplicaciones interactivas de visión por computador se tendrá una fuente de información representada por una cámara digital, por ejemplo, en robótica muchas veces se utilizan cámaras en los robots con un fin específico como distinguir formas y objetos. Para obtener imágenes desde una cámara se

utiliza la clase *VideoCapture* pero en lugar de indicar el nombre del archivo de video se envía el índice de la cámara a utilizar.

En la figura 8 se muestra un ejemplo de cómo capturar una fotografía y almacenarla en un archivo JPEG.

Figura 8. **Ejemplo de captura de fotografía desde una cámara**

```python
import cv2 as cv

#Se envía como parametro el índice de la cámara a utilizar
camCap = cv.VideoCapture(0)
flag, frame = camCap.read()

while(flag):
    cv.imshow('Imagen 1', frame)
    if cv.waitKey(1) & 0xFF == ord('t'):
        cv.imwrite('output/camera.jpg', frame)
        cv.destroyAllWindows()
        break
    flag, frame = camCap.read()

camCap.release()
```

Fuente: elaboración propia, empleando la herramienta https://carbon.now.sh.

Para este ejemplo se utiliza la función *waitKey(1)* el cual devuelve un número binario de 8 *bits* que representa un evento de teclado y se le aplica una máscara de 1’s para obtener su valor entero, esto se compara con el valor entero

de la letra ‘t’, indicando que cuando suceda un evento de teclado que presione la letra ‘t’ se cumplirá la condición y se capturara la imagen que se haya capturado en ese momento.

En la figura 9 se muestra un ejemplo de código sobre cómo capturar video desde una cámara y almacenarlo en un archivo.

Figura 9. **Ejemplo de captura de video desde una cámara**

```
import cv2 as cv

#Se inicializa el objeto capturador y se define la tasa de framse por segundo y la resolución
camCap = cv.VideoCapture(0)
fps = 30
size = (int(camCap.get(cv.CAP_PROP_FRAME_WIDTH)),
        int(camCap.get(cv.CAP_PROP_FRAME_HEIGHT)))
#Se inicialza un objeto de la clase VideoWriter el cual nos servira para almacenar el video
videoEscritura = cv.VideoWriter(
    'output/video2.avi', cv.VideoWriter_fourcc('I','4','2','0'), fps, size)
flag, frame = camCap.read()
numFrames = 300

while flag and numFrames > 0:
    videoEscritura.write(frame)
    flag, frame = camCap.read()
    numFrames -= 1
camCap.release()
```

Fuente: elaboración propia, empleando la herramienta https://carbon.now.sh.

1.4.2. Procesamiento de imágenes

Cuando se trabaja con imágenes, tarde o temprano, se necesitará aplicar modificaciones a estas, ya sea aplicando filtros, cambiando el tamaño o el color, recortar o agregar nuevo contenido a la imagen. A continuación, se presentan

algunas de las técnicas más utilizadas para el procesamiento de imágenes y como se implementa utilizando la librería *OpenCV* con el lenguaje de programación Python.

1.4.2.1. Conversión entre los distintos espacios de color

Un espacio de color es una forma de representar los colores de manera numérica, utilizando normalmente tres o cuatro valores. Para explicar este procedimiento es necesario presentar los espacios de color con los que se estará trabajando:

- Escala de grises, es un espacio de color que permite representar la información de los colores en una imagen como tonos de gris. Puede ser de mucha utilidad ya que utiliza un solo canal que representa la intensidad del color gris, haciendo que las operaciones en este espacio de color sean más rápidas y requieran de menos recursos de procesamiento.
- BGR, por sus siglas en inglés, espacio de color azul-verde-rojo, cada *pixel* de la imagen es representado como un arreglo de 3 elementos que representan la intensidad de cada uno de estos 3 colores en ese punto específico.
- HSV, espacio de color basado en los componentes de tinte, tonalidad y brillo.

En O*penCV* por defecto se maneja el espacio de color BGR, a partir de esto se presenta la forma de conversión a los espacios de color escala de grises y HSV.

La figura 10 muestra un ejemplo de código de la conversión desde el espacio de color BGR hacia una escala de grises, se aplican los conceptos anteriores para leer y guardar una imagen.

Figura 10. **Ejemplo de conversión a distintos espacios de color**

```python
import cv2 as cv

#Lectura de imagen
img = cv.imread('input/nasa.png')
#Conversion entre BGR<->Escala de grises
grises = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
#Conversion entre BGR<->HSV
hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)
#Mostar las imagenes convertidas en una ventana
cv.imshow('Escala de grises', grises)
cv.imshow('HSV', hsv)
#Almacenar imagenes
cv.imwrite('output/nasa_greyScale.png')
cv.imwrite('output/nasa_hsv.png'
#Esperar cualquier evento de teclado y cerrar todas las ventanas
cv.waitKey(0)
cv.destroyAllWindows()
```

Fuente: elaboración propia, empleando la herramienta https://carbon.now.sh.

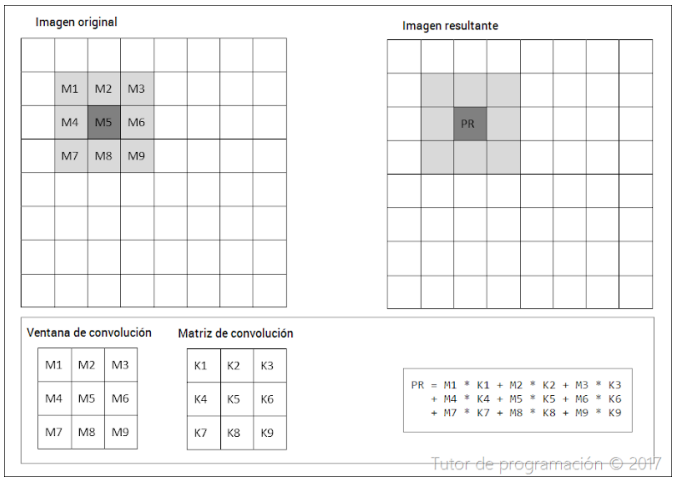
1.4.2.2. Aplicación de filtros

Un filtro de imagen es una operación que se aplica a una imagen para resaltar o mejorar alguna de sus características, para lograr esto se modifica la

matriz o matrices que componen la imagen aplicándole una determinada operación, en esta sección se aplicará la operación de matrices llamada convolución.

Para crear un filtro se requiere una matriz de convolución o *kernel*, típicamente será una matriz cuadrada de tamaños: 3x3, 5x5, 7x7, entre otros. Dependiendo de la función del filtrado que se quiera aplicar serán los valores de esta matriz de convolución. El filtro examina, sucesivamente, cada *pixel* de la imagen. Para calcular el valor de un *pixel* tomamos los elementos vecinos al mismo y creamos una ventana de igual tamaño que la matriz de convolución, luego multiplicamos las matrices elemento por elemento y sumamos los resultados para obtener el pixel deseado, normalmente se toma el que se ubica en el centro. En la figura 11 se muestra una gráfica de la aplicación de un filtro a una imagen.

Figura 11. **Aplicación de un filtro 3x3**



Fuente: ABREGO, Carmelo. *Filtro 3x3.* http://acodigo.blogspot.com/2017/05/filtros-de-imagenes-por-convolucion-de.html. Consulta: 16 de febrero de 2020.

En OpenCV se puede aplicar filtros a imágenes usando *kernels* de dos dimensiones mediante la función *filter2D()*, en la Figura 12 se observa un ejemplo de la aplicación de un filtro 3x3 a una imagen.

Figura 12. **Ejemplo de aplicación de filtros**

```python
import cv2 as cv
import numpy as np

#Definimos el kernel 3x3
kernel = np.array([[-1, 0, 1],
                   [-1, 0, 1],
                   [-1, 0, 1]])
img = cv.imread('input/opencv_logo.png')
#Convertimos la imagen al espacio de color escala de grises
img_gs = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
#Aplicación del filtro
filter_img = cv.filter2D(img_gs, -1, kernel)
#Mostrar una sola imagen en pantalla
images = np.hstack((img_gs, filter_img))
cv.imshow('Resultado', images)

cv.waitKey(0)
cv.destroyAllWindows()
```

Fuente: elaboración propia, empleando la herramienta https://carbon.now.sh.

En la figura 13 se muestra el resultado de este fragmento de código aplicado a la imagen del logo de OpenCV.

Figura 13. **Resultado de aplicar un filtro a una imagen**

Fuente: elaboración propia, empleando *Python* y *OpenCV*.

1.4.2.3. Detección de bordes

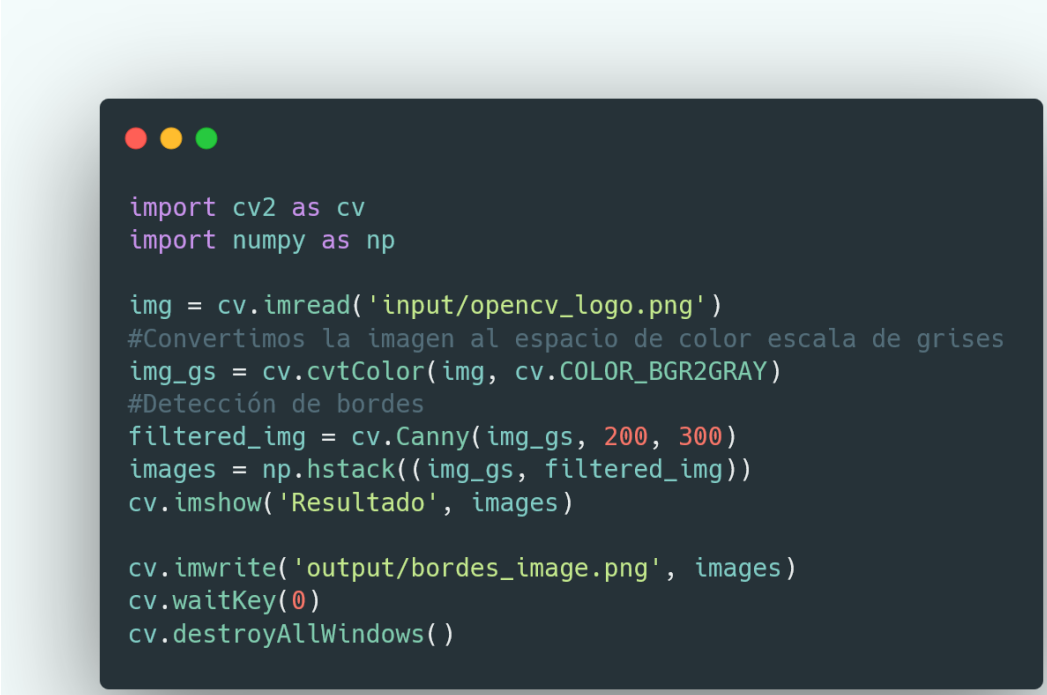
Los bordes de una imagen digital pueden ser definidos como transiciones entre dos regiones de distinto color. Esta característica de la imagen proporciona información muy valiosa acerca de las fronteras entre los objetos y se puede utilizar para segmentar la imagen y reconocer objetos en ella.

La mayoría de las técnicas para detectar bordes se aplican en una imagen que tiene definido el espacio de color escala en grises. *OpenCV* ofrece muchas funciones para detectar bordes en las imágenes, puede ser aplicando un filtro a la imagen como se indicó anteriormente o implementando un algoritmo conocido para este propósito.

Canny() es una función de OpenCV que sirve para detectar bordes en una imagen, es una implementación del algoritmo creado por John F. Canny para este propósito.

A continuación en la figura 14 se muestra un ejemplo de cómo aplicar la función *Canny()* a una imagen y en la figura 15 se muestra el resultado que se obtiene de aplicar este fragmento de código.

Figura 14. **Ejemplo detección de bordes utilizando función *Canny()***



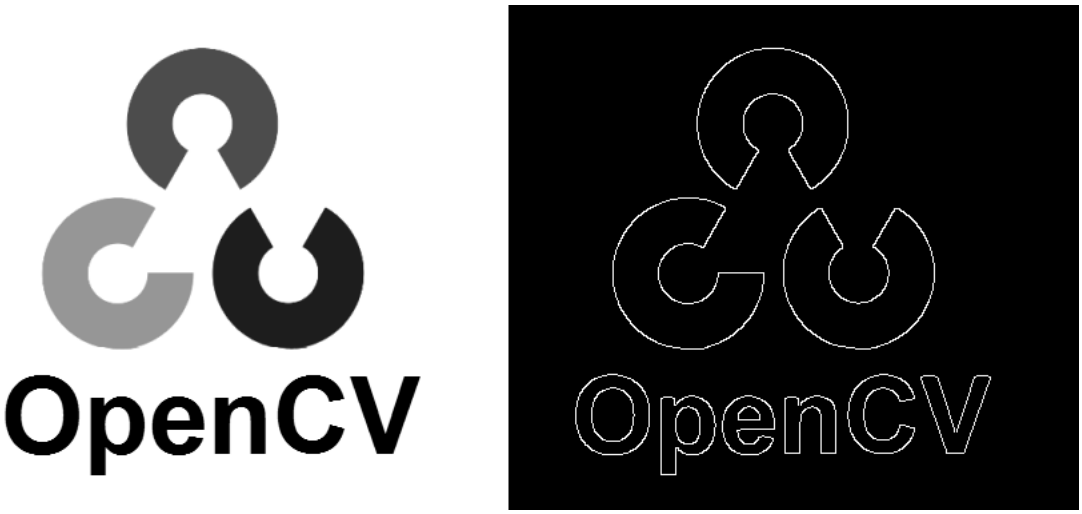
```
import cv2 as cv
import numpy as np

img = cv.imread('input/opencv_logo.png')
#Convertimos la imagen al espacio de color escala de grises
img_gs = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
#Detección de bordes
filtered_img = cv.Canny(img_gs, 200, 300)
images = np.hstack((img_gs, filtered_img))
cv.imshow('Resultado', images)

cv.imwrite('output/bordes_image.png', images)
cv.waitKey(0)
cv.destroyAllWindows()
```

Fuente: elaboración propia, empleando la herramienta https://carbon.now.sh.

Figura 15. **Resultado de aplicar la función *Canny()* a una imagen**



Fuente: elaboración propia, empleando *Python* y *OpenCV.*

1.4.2.4. Detección de contornos

Un contorno se puede describir como una curva que une todos los puntos continuos, que tienen el mismo color o intensidad en una imagen. Los contornos son una herramienta útil para el análisis de formas, detección y reconocimiento de objetos.

En *OpenCV* se puede utilizar la función *findContours()* que ayuda a extraer los contornos de una imagen. Funciona mejor en imágenes binarias y se recomienda cambiar a un espacio de color en escala de grises y aplicar filtros o algoritmos para detectar bordes en la imagen y así obtener una imagen binaria. El resultado de esta función es una lista que contiene todos los contornos de la imagen, asimismo, se utiliza la función *drawContours()* que toma este listado y agrega el contorno con el color que especifiquemos a la imagen donde se aplicó la función.

En la figura 16 se mostrará un fragmento de código que se puede utilizar para reconocer contornos en una imagen utilizando *OpenCV.*

Figura 16. **Ejemplo detección de contornos utilizando la función *findContours()***

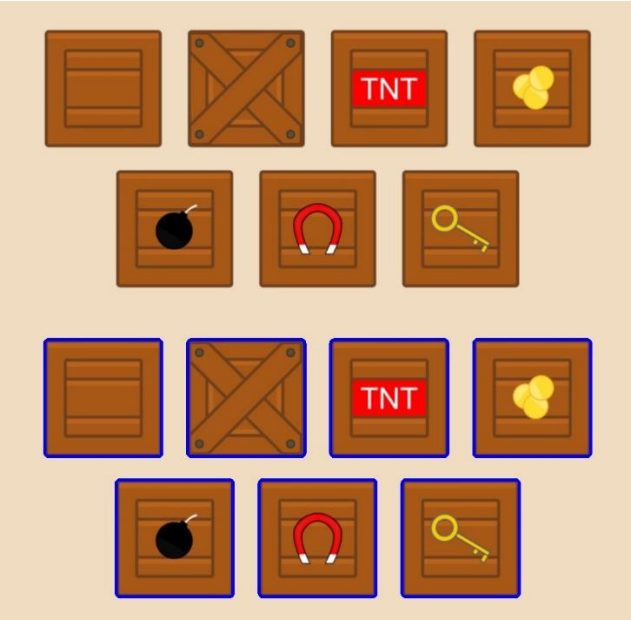
```python
import cv2 as cv
import numpy as np

img = cv.imread('input/boxes2d.png')
#Convertmos la imagen al espacio de color escala de grises
img_gs = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
#Detección de bordes
edges = cv.Canny(img_gs, 200, 300)
#Detección de contornos
contours, hierarchy = cv.findContours(edges,
    cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
#Dibujar los contornos encontrados con color azul
img = cv.drawContours(img, contours, -1, (255,0,0), 3)
#Guardar la imagen con los contornos
cv.imshow('contornos', img)
cv.imwrite('contornos.jpg', img)
cv.waitKey(0)
cv.destroyAllWindows()
```

Fuente: elaboración propia, empleando la herramienta https://carbon.now.sh.

En la figura 17 se observa el resultado de ejecutar este fragmento de código.

Figura 17. **Resultado de aplicar las funciones *findContours()* y *drawContours()* a una imagen**



Fuente: elaboración propia, empleando *Python* y *OpenCV.*

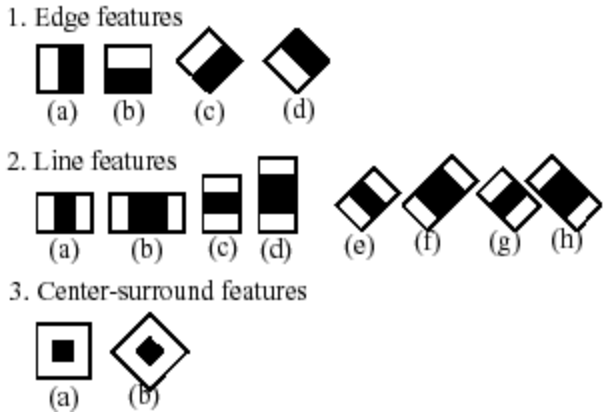
1.4.3. Reconocimiento de objetos

Detectar un objeto es la capacidad de concluir si en cierta región de una imagen hay un objeto no identificado, utilizando las técnicas de las secciones anteriores se puede llevar a cabo esta acción. Reconocer se refiere a la capacidad de identificar el objeto. Por ejemplo, en aplicaciones de reconocimiento facial primero debemos detectar el área donde exista una cara y luego aplicar las técnicas necesarias para reconocer si es una cara que permita ejecutar cierta acción.

1.4.3.1. Clasificador *Haar cascade*

Es un algoritmo de aprendizaje automático utilizado para reconocer objetos en una imagen. El algoritmo extrae las características más comunes del objeto que se quiere reconocer, para esto utiliza regiones rectangulares adyacentes en una ubicación específica, calcula la suma de la intensidad de los pixeles en cada región y obtiene la diferencia entre estas.

Figura 18. **Características comunes en un clasificador *Haar cascade***



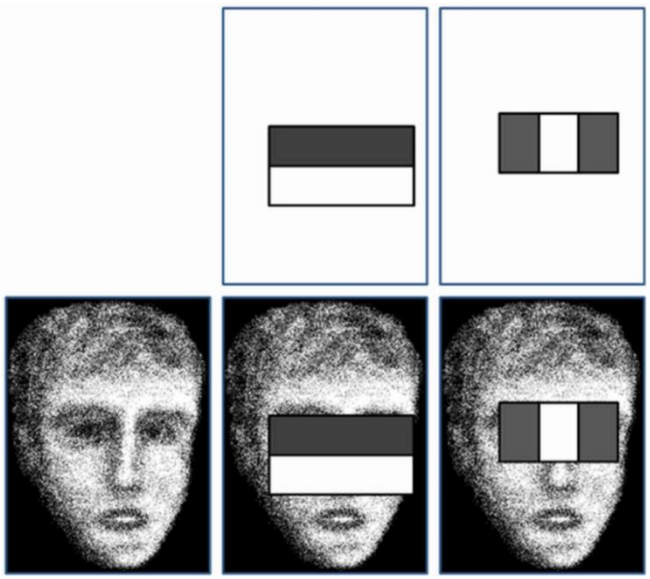
Fuente: Opencv dev team. *Cascade classification.* https://docs.opencv.org/2.4/_images/haarfeatures.png. Consulta: 18 de febrero de 2020.

Considerando el resultado de buscar estas características en una imagen, se podría seleccionar una característica que nos ayude a identificar los ojos de una persona, dado que, normalmente la región de los ojos es más obscura que la región de las mejillas o la nariz como se observa en el cuadro 2 de la fila 2 en la figura 19; se puede seleccionar otra característica que nos ayude a identificar la nariz, dado que, normalmente el área entre los ojos es más oscura que el puente que se forma en el tabique nasal como se observa en el cuadro 3 de la

fila 2 en la figura 19. Sin embargo, estas dos características resultarían irrelevantes si se aplicaran en cualquier otra región del rostro.

Dado que una característica resulta útil para distinguir una región en particular del objeto, se deben seleccionar aquellas características que al unirse puedan formar un clasificador capaz de identificar un conjunto de características propias del objeto a reconocer, de esa manera se generan varios clasificadores con distintas características de la imagen. El término *cascade*, cascada, en el nombre del clasificador se refiere a que se utilizan los clasificadores creados en forma de estaciones para identificar todas las características del objeto de interés y obtener un resultado fiable que responde a la siguiente pregunta: ¿El objeto X está presente en la siguiente imagen?

Figura 19. **Características encontradas en una imagen**



Fuente: SOYATA, *Tolga. Face Recognition: A Tutorial on Computational Aspects.* Consulta: 18 de febrero de 2020.

1.4.3.1.1. Creación de *Haar cascade* a partir de un conjunto de imágenes

OpenCV incluye una serie de herramientas que sirven para entrenar un clasificador de este tipo, se detalla una serie de pasos para la creación de este clasificador.

El primer paso es coleccionar un conjunto de imágenes y ordenarlas de la siguiente forma: en la carpeta positivas se deben incluir todas las imágenes que contienen el objeto a clasificar, deben ser de dimensión parecida y se debe recortar la imagen a modo que solo se observe el objeto. En la carpeta negativas se deben agregar todas aquellas imágenes que no contengan el objeto a clasificar. En la tabla I se muestra un ejemplo de la estructura de la carpeta donde se entrene el clasificador.

Tabla I. **Estructura de los ficheros y carpetas para entrenar el clasificador**

positivas/
imagen1.jpg
imagen2.jpg
bg.txt
negativas/
imagen1.jpg
imagen2.jpg
bg.txt

Fuente: elaboración propia, empleando Microsoft Word.

El segundo paso es crear un conjunto de ejemplos con las imágenes que contienen al objeto, este conjunto es creado a partir de aplicar transformaciones rotacionales aleatorias a todas las imágenes en la carpeta positivas utilizando el comando que se muestra en la tabla II.

Tabla II. **Creación de ejemplos a partir de las imágenes que contienen el objeto a reconocer**

opencv_createsamples –img ~/positivas –bg ~/positivas/bg.txt –vec ~/trainvec.vc -pngoutput -maxxangle 0.1 -maxyangle 0.1 -maxzangle 0.1

Fuente: elaboración propia, empleando Microsoft Word.

El resultado del comando anterior será el archivo *trainvec.vec* que enumeran las características comunes de todas las imágenes que contienen al objeto que se intentará reconocer.

El último paso es entrenar el clasificador, *OpenCV* proporciona el comando *opencv_traincascade* en la tabla III se muestra la forma en que se debe utilizar el comando para entrenar el clasificador y los parámetros que se utilizan.

Tabla III. **Comando utilizado para entrenar un clasificador**

opencv_traincascade –data clasificador.xml –vec ~/trainvec.vec

Fuente: elaboración propia, empleando Microsoft Word.

El resultado será el archivo clasificador.xml que contiene la información de todas las estaciones y características encontradas para reconocer al objeto.

1.4.3.1.2. Ejemplo de clasificador

A modo de ejemplo se utilizará un clasificador entrenado para reconocer la señal de transito *stop*, el clasificador se entrenó a partir de un conjunto de 50 imágenes que contienen la señal *stop*; y 600 imágenes que no contienen la señal mencionada.

En la figura 20 se mostrará un fragmento de código que se puede utilizar para reconocer la señal *stop* utilizando un clasificador del tipo *Haar-Cascade* en *OpenCV*, posteriormente en la figura 21 se muestra el resultado de aplicar este clasificador a una señal de *stop* elaborada para fines de este trabajo.

Figura 20. **Ejemplo de reconocimiento de señal de transito utilizando clasificador *Haar-cascade***

```python
import cv2 as cv
import numpy as np

# Carga de clasificador a partir de archivo xml
clasificador = cv.CascadeClassifier('input/clasificador.xml')
# Imagen donde se encuentra el objeto
img = cv.imread('input/h3.bmp')
# Cambiamos de espacio de color a escala de grises
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
# objetos contiene una lista de rectangulos en donde se reconocio el objeto
objetos = clasificador.detectMultiScale(gray, 1.1, 5)
for (x,y,w,h) in objetos:
    # Dibujamos un rectangulo de color azul delimitando el objeto
    img = cv.rectangle(img, (x,y), (x+w, y+h), (255,0,0),2)

cv.imshow('img', img)
cv.imwrite('output/signalDetected.jpg', img)
cv.waitKey(0)
cv.destroyAllWindows()
```

Fuente: elaboración propia, empleando la herramienta https://carbon.now.sh.

Figura 21. **Resultado de aplicar el clasificador de señal de tránsito *stop* tipo Haar-Cascade**

Fuente: elaboración propia, empleando cámara fotográfica y software *OpenCV.*

2. CONFIGURACIÓN DEL HARDWARE

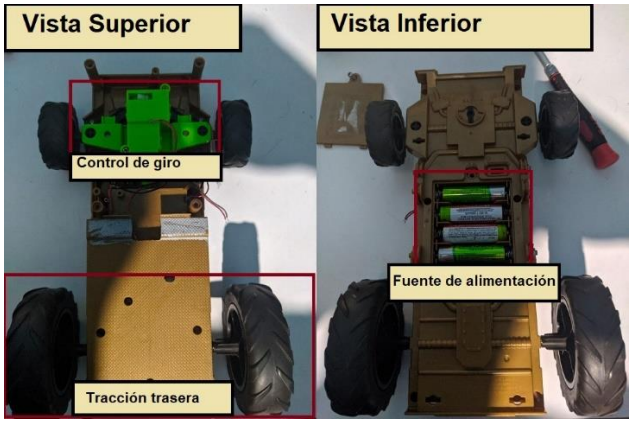
2.1. Estructura física del prototipo

Para el desarrollo del presente trabajo se reciclo un vehículo a escala de radio control, los puntos que se consideraron para la selección del vehículo a escala fueron los siguientes:

- Eje dedicado a la tracción del vehículo que fuese controlado por motor eléctrico de corriente directa.
- Eje de dirección que fuese controlado por motor eléctrico de corriente directa.
- Poseer espacio propio para incluir fuente de alimentación.

En la figura 22 se observa el vehículo a escala seleccionado y se detallan las partes importantes.

Figura 22. **Estructura del vehículo a escala**



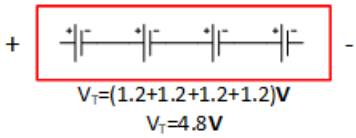
Fuente: elaboración propia, empleando programa para edición de imágenes *Paint*.

2.2. Fuente de alimentación

La fuente de alimentación utilizada para energizar la placa de control de motores del vehículo a escala se compone de 4 baterías “AA” de 1,2 Volts cada una.

En la figura 23 se muestra un diagrama eléctrico de la fuente de alimentación que consta de 4 fuentes independientes conectadas en serie.

Figura 23. **Fuentes en serie**



Fuente: elaboración propia, empleando Microsoft Visio.

2.3. Placa de control de motores

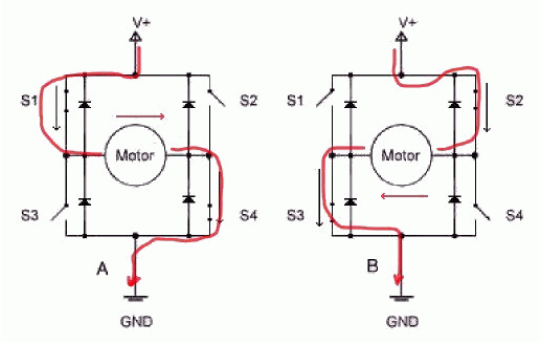
Una placa de control de motores es un arreglo de componentes eléctricos y electrónicos que son capaces de controlar las características principales de un motor, ya sea de corriente directa o alterna, tales como la velocidad y dirección de giro.

2.3.1. Puente H L298N

Un puente H es un circuito electrónico que se utiliza comúnmente para permitir que un motor de corriente directa pueda girar en ambos sentidos. En la figura 24 se observa el funcionamiento de este circuito, el giro de un motor se

puede alterar cuando se cambia la trayectoria del flujo de la corriente en sus bobinas.

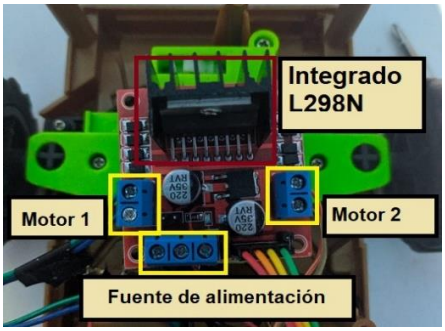
Figura 24. **Funcionamiento Puente H**



Fuente: GARCÍA, Victor. *El Puente H.* https://www.hispavila.com/el-puente-h/. Consulta: 24 de febrero de 2020.

Este módulo utiliza un integrado que contiene 2 configuraciones de Puente H y por lo tanto permite controlar la velocidad y dirección de dos motores de corriente directa. En la figura 25 se observa el módulo L 298N como parte de la estructura física del vehículo a escala mostrado en la sección anterior.

Figura 25. **Adaptación de módulo L 298N a estructura física del vehículo a escala**

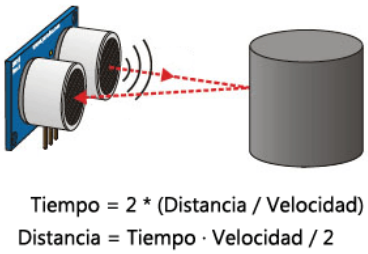


Fuente: elaboración propia, empleando programa para edición de imágenes *Paint.*

2.4. Sensor de distancia

El principio de funcionamiento de un sensor de distancia se basa en la propiedad de reflexión de las señales, típicamente se utiliza una señal de audio de muy alta frecuencia conocida como señal ultrasónica; el sensor tiene un arreglo de emisor-receptor como se muestra en la figura 26, el emisor envía una señal ultrasónica por el espacio y cuando esta es reflejada por un objeto se dirige al receptor, para calcular la distancia se utiliza la ecuación del movimiento rectilíneo uniforme dado que la velocidad del sonido es una constante y se puede cuantificar el tiempo que le tomo a la señal acústica en regresar al receptor.

Figura 26. **Emisor-Receptor de señal ultrasónica**



Fuente: LLAMAS, Luis. *Sensor ultrasónico HC-SR04*. https://tecnopatafisica.com/tecno3eso/teoria/robotica/27-hcsr04. Consulta: 24 de febrero de 2020.

2.4.1. Módulo HC-SR04

Es un módulo de emisor y receptor de señales ultrasónicas que trabaja a una frecuencia de 40 kHz, como se observa en la figura 27 el módulo cuenta con 4 pines, alimentación Vcc, un pin de tierra o masa GND, un disparador Trig que activa el emisor de señales ultrasónicas emitiendo pulsos y un receptor *Echo* que detecta las señales reflejadas. Cuando el disparador emite el tren de señales

ultrasónicas se mide el tiempo que le toma al receptor recibir la señal reflejada y así se puede calcular la distancia a la que se encuentra el objeto reflector.

Figura 27. **Módulo HC-SR04**

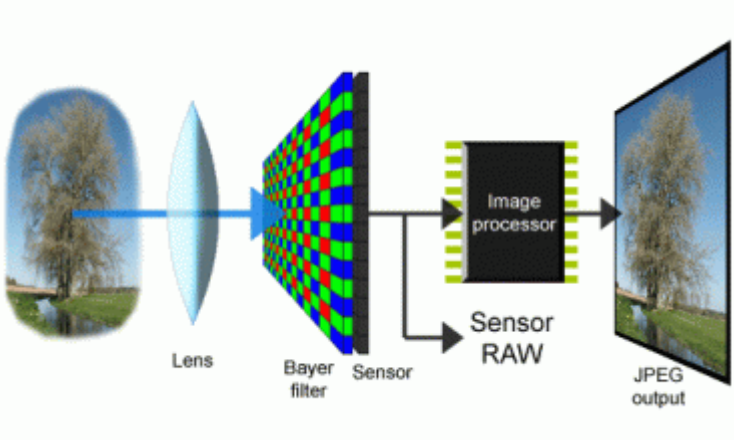
Fuente: Web-robótica. *Módulo ultrasónico HC-SR04.* https://www.web-robotica.com/arduino/como-usar-el-sensor-de-distancia-ultrasonico-hc-sr04-con-arduino. Consulta: 24 de febrero de 2020.

2.5. Cámara digital

Una cámara digital es una matriz de sensores que permiten captar la intensidad de color que perciben para formar una imagen, utiliza componentes electrónicos capaces de procesar la imagen y hacen posible almacenar esta información en un formato digital. A cada uno de estos sensores se le conoce como *pixel* y comúnmente mientras más *pixeles* tenga el sensor mejor será la calidad de la imagen.

En la figura 28 se puede observar a grandes rasgos el funcionamiento de una cámara digital.

Figura 28. **Funcionamiento de cámara digital**



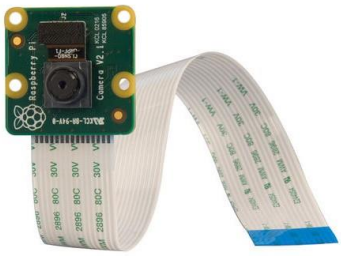
Fuente: NICOLLE, Cynthia. *Cámara digital*. https://cinthyanicolle901.wordpress.com/2015/04/13/caracteristicas-de-una-camara-digital/. Consulta: 25 de febrero de 2020.

2.5.1. Raspberry Pi camera v1

Es un módulo de cámara digital de 5 megapíxeles diseñado específicamente para esta tarjeta de desarrollo. Tiene un conector especial diseñado para la conexión de cámaras.

En la figura 29 se muestra el módulo de cámara digital.

Figura 29. **Módulo de cámara para *Raspberry Pi***



Fuente: Bricogeek. *Cámara Raspberry Pi*. https://tienda.bricogeek.com/accesorios-raspberry-pi/822-camara-raspberry-pi-v2-8-megapixels.html. Consulta: 25 de febrero de 2020.

2.6. Pista modelo

Para propósito de este trabajo se elaboró una pequeña pista con la escala apropiada del vehículo para la toma de datos y realización de todas las pruebas de conducción autónoma.

2.6.1. Pista

Es un espacio físico con características cambiantes, cambios de dirección y objetos aledaños, que le otorgan información al vehículo a escala para la toma de decisiones, es de un solo carril y este se delimita de ambos lados utilizando una cinta negra.

Figura 30. **Pista a escala**

Fuente: elaboración propia, empleando cámara fotográfica.

2.6.2. Señales de tránsito

Se elaboró 3 señales de *stop* o alto a escala. Las señales de tránsito se pueden colocar en cualquier lugar de la pista.

Figura 31. **Señal de *stop* ubicada en un punto de la pista**

Fuente: elaboración propia, empleando cámara fotográfica.

2.6.3. Obstáculos

Un obstáculo se puede definir como cualquier objeto que no permita la circulación del vehículo por un tramo de la pista, por ejemplo, un bebé humano como se observa en la figura 32.

Figura 32. **Obstáculo en la pista**

Fuente: Eeaboración propia, empleando cámara fotográfica.

La pista y las señales de tránsito se elaboraron de tal forma que puedan cambiar de posición fácilmente, es decir, la pista modelo no tiene un diseño especifico se puede adaptar a la forma que se le quiera dar. Por ejemplo, se puede hacer una pista recta, un *zig-zag*, un cuadrado, entre otros. Las señales de tránsito tienen una base que las soporta sin necesidad de estar pegadas a una superficie.

Estas características hacen posible la creación de distintas pistas que permite obtener un conjunto de datos más grande para entrenar el modelo que permitirá la conducción en modo autónomo del vehículo a escala.

3. CONFIGURACIÓN DEL SOFTWARE

3.1. Tarjeta de desarrollo

Para el desarrollo del presente trabajo de graduación se seleccionó un ordenador de placa reducida llamado *Raspberry Pi,* específicamente la versión del hardware 3B+.

3.1.1. Raspberry Pi

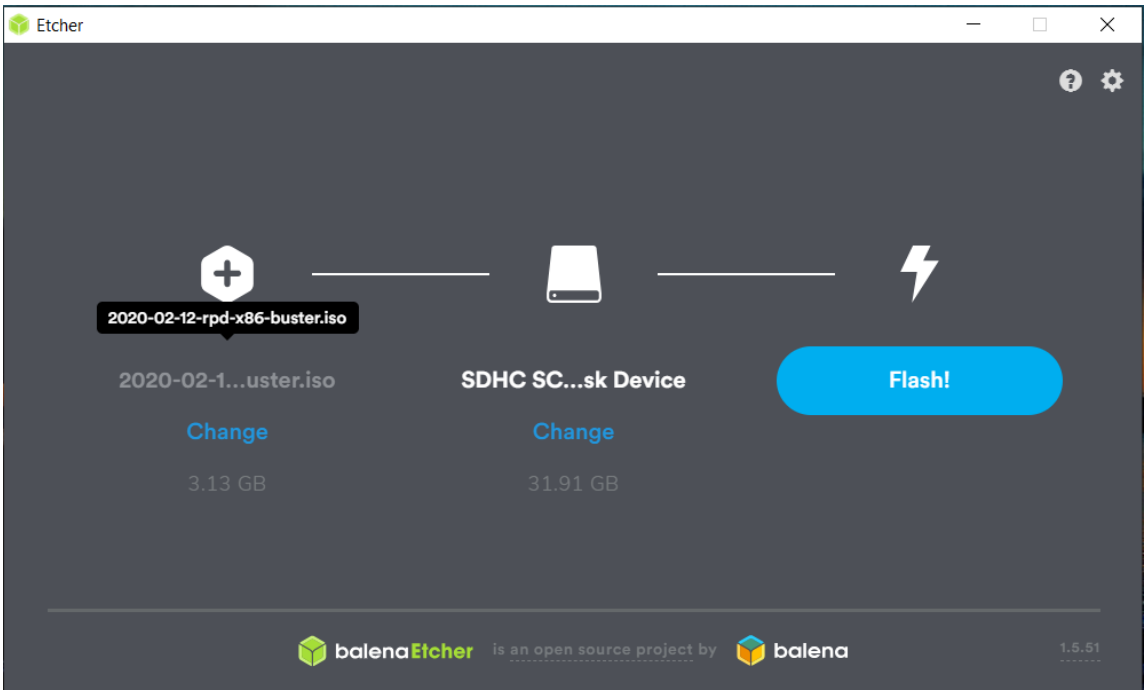
Este ordenador de placa reducida es capaz de ejecutar distintos sistemas operativos, para propósito de este trabajo se detallará como instalar una versión del sistema operativo *Raspbian,* que es un sistema de código abierto y esta optimizado para el hardware de la *Raspberry Pi.*

3.1.1.1. Instalación de Raspbian

Se debe descargar la imagen del sistema operativo desde la página oficial de *Raspbian* o *Raspberry Pi*, se debe escribir la imagen descargada en la tarjeta *micro sd* que será el disco de arranque. Esta acción se realiza utilizando el programa de código abierto llamado *balenaEtcher*, dentro de la aplicación se selecciona la imagen descargada en formato *.zip* y el nombre de la tarjeta micro sd que se utiliza, luego se selecciona la opción *Flash*. Esta operación puede durar unos minutos.

En la figura 33 se observa el programa utilizado para escribir la imagen del sistema operativo en la tarjeta micro sd.

Figura 33. **Escritura de Sistema Operativo en tarjeta micro sd**



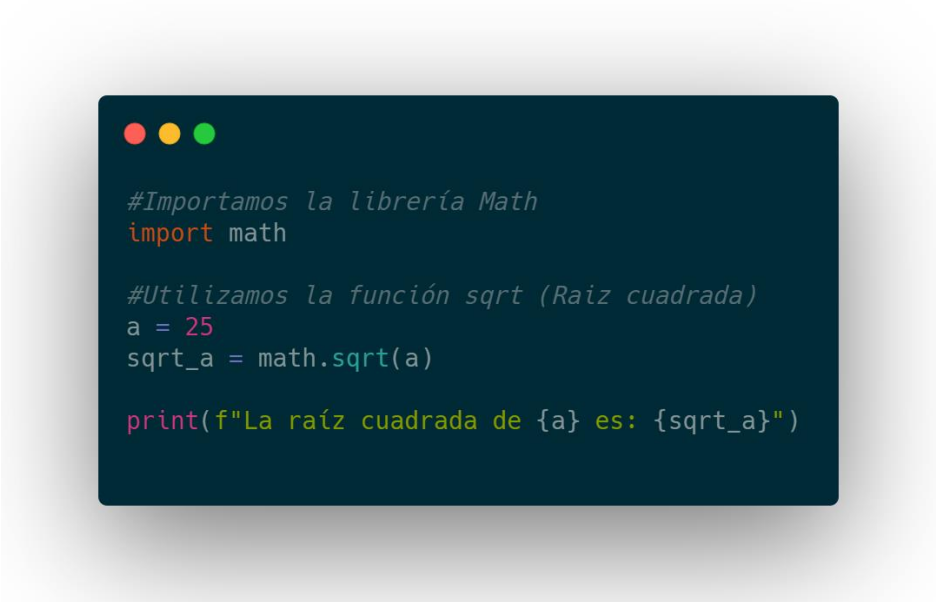
Fuente: elaboración propia, utilizando herramienta de recortes de mi computador personal.

3.2. Instalación de librerías

Una librería es un conjunto de datos y código que nos permite utilizar algoritmos o procedimientos de software que tengan un fin específico. Por ejemplo, la librería *math* en el lenguaje de programación *Python* nos permite acceder a operaciones matemáticas complejas llamando a la función por su nombre.

En la figura 34 se muestra un ejemplo de cómo utilizar la librería *math* en el lenguaje de programación *Python.*

Figura 34. **Utilización de una librería de código**



```python
#Importamos la librería Math
import math

#Utilizamos la función sqrt (Raiz cuadrada)
a = 25
sqrt_a = math.sqrt(a)

print(f"La raíz cuadrada de {a} es: {sqrt_a}")
```

Fuente: elaboración propia, empleando la herramienta https://carbon.now.sh.

3.2.1. Instalación de *OpenCV*

Para agregar la librería *OpenCV* se utilizó la herramienta de instalación de paquetes del lenguaje de programación *Python*. Utilizando dicha herramienta se puede instalar una versión no oficial pre-compilada de los paquetes utilizados por *OpenCV*, cabe resaltar que no es una versión sostenida por los desarrolladores de la librería oficial sino fue creada por la comunidad para lograr una instalación más sencilla.

En la tabla IV se detallan de forma secuencial los comandos necesarios para la instalación de la librería utilizando la herramienta *pip* por sus siglas en inglés, *package installer for Python*, iniciando propiamente con la instalación de esta herramienta para luego instalar el paquete *OpenCV.*

Tabla IV. **Instalación de *OpenCV***

Instalación de dependencias a nivel de Sistema operativo
$ sudo apt-get install libhdf5-dev libhdf5-serial-dev libhdf5-100
$ sudo apt-get install libqtgui4 libqtwebkit4 libqt4-test python3-pyqt5
$ sudo apt-get install libatlas-base-dev
$ sudo apt-get install libjasper-dev
Paso 1: Descargar la herramienta “pip” desde el repositorio de *python*
pi@raspberrypi:~$ wget https://bootstrap.pypa.io/get-pip.py
Paso 2: Instalar la herramienta “pip” utilizando *pyhton 3*
pi@raspberrypi:~$ sudo python3 get-pip.py
Paso 3: Instalar el paquete de computación científica *numpy*
pi@raspberrypi:~$ sudo pip install numpy
Paso 4: Instalar la librería O*penCV*
pi@raspberrypi:~$ sudo pip install opencv-contrib-python

Fuente: elaboración propia, empleando Microsoft Word.

3.2.1.1. Instalación desde código fuente

Este tipo de instalación se refiere al proceso de compilación del código fuente de un paquete de software en específico, el código fuente de *OpenCV* es catalogado como software libre y puede ser utilizado por cualquier persona. Una de las ventajas de este tipo de instalación es que siempre se manejará la versión estable más reciente, dependiendo de nuestras intenciones y conocimiento también se podrá instalar versiones en fase de pruebas mejor conocidas como *beta*.

En la tabla V se indica de manera informal una serie de pasos a seguir para la instalación de *OpenCV* desde su código fuente.

Tabla V. **Instalación de *OpenCV* desde código fuente**

Paso 1: Actualizar los paquetes existentes del sistema
$ sudo apt-get update && sudo apt-get upgrade
Paso 2: Instalar las herramientas necesarias para la compilación, paquetes para manejo de distintos tipos de imágenes, códecs de video y versión de Python de desarrollo.
$ sudo apt-get install build-essential cmake pkg-config
$ sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng-dev
$ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
$ sudo apt-get install libxvidcore-dev libx264-dev
$ sudo apt-get install python3-dev
Paso 3: Descargar la versión requerida de *OpenCV* desde su repositorio oficial, reemplazar la "X" por su versión deseada
$ wget -O opencv.zip https://github.com/opencv/opencv/archive/X.X.X.zip
$ wget -O opencv_contrib.zip https://github.com/opencv/opencv_contrib/archive/X.X.X.zip
$ unzip opencv.zip
$ unzip opencv_contrib.zip
$ mv opencv-X.X.X opencv
$ mv opencv_contrib-X.X.X opencv_contrib
Paso 4: Configurar la compilación del código
$ cd ~/opencv
$ mkdir build
$ cd build
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \
-D CMAKE_INSTALL_PREFIX=/usr/local \
-D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules \
-D ENABLE_NEON=ON \
-D ENABLE_VFPV3=ON \
-D BUILD_TESTS=OFF \
-D INSTALL_PYTHON_EXAMPLES=ON \
-D OPENCV_ENABLE_NONFREE=ON \
-D CMAKE_SHARED_LINKER_FLAGS=-latomic \
-D BUILD_EXAMPLES=OFF ..
Paso 5: Compilación del código fuente
$ make -j4
Paso 6: Si los pasos anteriores fueron exitosos, se concluye que el código compilo con éxito y se puede proceder con la instalación
$ sudo make install
$ sudo ldconfig

Fuente: elaboración propia, empleando Microsoft Word.

3.3. Configuración de periféricos

Un periférico es un dispositivo de hardware que se conecta a un ordenador y le permite comunicarse con el mundo exterior. Por ejemplo, un micrófono permite convertir las ondas de sonido en señales eléctricas digitales que el ordenador puede manipular.

Un puerto es una interfaz que se utiliza para comunicar distintos elementos de hardware o software para envío y recepción de información. Un periférico se conecta al ordenador utilizando un puerto.

El ordenador de placa reducida *Raspberry Pi* cuenta con distintos puertos disponibles, entre los cuales destacan:

- 4 puertos *USB* 2.0
- *Wi-Fi* + *Bluetooth*
- *Ethernet*
- Puerto CSI para conectar una cámara
- *GPIO*, pines de entrada y salida de propósito general

3.3.1. Cámara

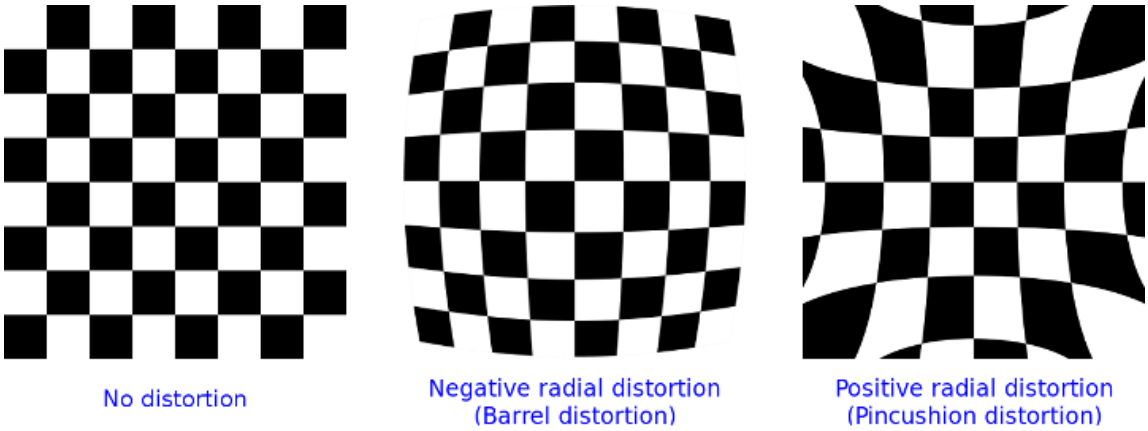
Existen varias formas de conectar una cámara a un *Raspberry Pi*, mediante un puerto *USB* o utilizando el puerto CSI para el desarrollo de este trabajo se utiliza el puerto mencionado mediante el cual se conecta una cámara digital diseñada para el ordenador de placa reducida *Raspberry Pi.*

3.3.1.1. Calibración

Es común que la cámara introduzca cierto efecto de distorsión en las imágenes. Existen dos tipos principales de distorsión, la distorsión radial y la distorsión tangencial.

La distorsión radial hace que las líneas rectas parezcan curvas, se hace más notable cuando las líneas rectas están lejos del centro de la imagen.

Figura 35. **Distorsión radial**



Fuente: Radial distortion, *OpenCV camera calibration*.
https://docs.opencv.org/master/distortion_examples.png. Consulta: 23 de julio de 2020.

De forma similar, la distorsión tangencial se produce cuando la lente no está alineada perfectamente paralela al plano de la imagen. Por lo tanto, algunas áreas de la imagen pueden verse más cerca de lo esperado.

El proceso de calibración consiste en encontrar la matriz de parámetros de la cámara que sean capaces de corregir las distorsiones antes mencionadas. La librería *OpenCV* implementa una función llamada *cameraCalibration* que permite

obtener una matriz de parámetros optimizados para eliminar la distorsión en las imágenes.

3.3.2. Utilización de GPIO

Los pines de entrada y salida de propósito general de la *Raspberry Pi* pueden ser configurados mediante software, utilizando una interfaz de configuración en el lenguaje de programación preferido.

Para la configuración de los pines se utilizó la librería *RPi.GPIO* creada específicamente para manejar los pines *GPIO* de la *Raspberry Pi* utilizando el lenguaje de programación *Python.* En esta librería hay 2 modos para definir la numeración de los pines:

- *BOARD*: se refiere a la numeración de los pines tal cual se encuentran físicamente en la tarjeta de desarrollo, su ventaja es que funciona independiente de la versión de la tarjeta.
- *BCM:* hace referencia al número de canal configurado en el *SOC, System on Chip,* para este enfoque se debe utilizar la hoja de datos del fabricante y puede variar dependiendo de la versión de la tarjeta de desarrollo.

En la tabla VI, se muestra como configurar los pines *GPIO* como entrada y salida utilizando la librería *RPI.GPIO*.

Tabla VI. **Utilización de *GPIO***

```
# Importar la librería RPi.GPIO y verificar que no haya ningún error en este paso
try:
   import RPi.GPIO as GPIO
except RuntimeError:
   print("Error al importar. Validar los privilegios de acceso a GPIO")
# Definir la numeración de los pines.
GPIO.setmode(GPIO.BCM)
# Configurar el pin 6 como salida
gpio.setup(6, gpio.OUT)
# Configurar el pin 6 en alto
gpio.setup(6, GPIO.HIGH)
# Configurar el pin e en bajo
gpio.setup(6, GPIO.HIGH)
# Configurar el pin 13 como entrada y configurar una resistencia pull-up o pull-down
gpio.setup(13, gpio.IN, pull_up_down=GPIO.PUD_UP)
# Leer el estado del pin 13
If GPIO.input(13):
   print('Pin 13 tiene una entrada de 3.3V')
else:
   print('Pin 13 tiene una entrada de 0V')
# Por último se deben limpiar las configuraciones actuales de los pines
GPIO.cleanup()
```

Fuente: elaboración propia, empleando Microsoft Word.

4. IMPLEMENTACIÓN DE UN PROTOTIPO DE VEHÍCULO AUTÓNOMO

4.1. Obtención de datos

Utilizando el lenguaje de programación *Python* se creó una interfaz que fuera capaz de controlar el hardware e interpretar los datos obtenidos del sensor de cámara en la tarjeta de desarrollo *Raspberry Pi.*

4.1.1. Captura de imágenes y dirección de movimiento

El conjunto de datos se obtiene a partir de las imágenes de entorno capturadas desde la cámara integrada en el vehículo a escala, para llevar a cabo esta parte se implementó un programa en el lenguaje de programación *Python* que define cada cuánto tiempo se debe tomar una fotografía y su clasificación; la clasificación se basa en la dirección de movimiento definida por el usuario al momento de tomar la fotografía.

Se definió una estructura de ficheros que contiene las imágenes clasificadas por su dirección de movimiento, el programa de *Python* se encarga de tomar la fotografía y almacenarla en la carpeta destinada a la dirección de movimiento específica.

En la figura 36 se puede observar la estructura de los ficheros y una muestra de las imágenes clasificadas según su dirección de movimiento.

Figura 36. **Captura de imágenes y dirección de movimiento**

Fuente: elaboración propia, empleando programa para edición de imágenes *Paint*

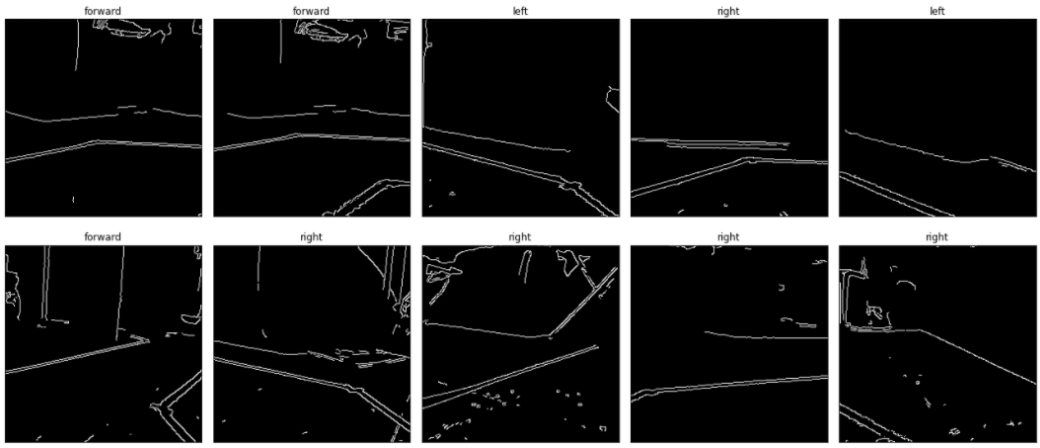
4.1.2. Procesamiento de imágenes

Las técnicas de procesamiento de imágenes utilizadas previo al entrenamiento del modelo computacional se detallan a continuación:

- Conversión al espacio de color escala de grises.
- Cambiar el tamaño de las imágenes a 240x240 pixeles de ancho y alto.
- Obtener los bordes de la imagen utilizando el algoritmo *Canny* para detección de bordes.
- Convertir la imagen a un arreglo dimensional de 240*240 unidades.

En la figura 37 se observa una muestra de un conjunto de imágenes aleatorias luego de aplicar las técnicas de procesamiento propuestas, en la parte superior de cada imagen se muestra la categoría de movimiento con la que se clasifico.

Figura 37. **Muestra de imágenes posterior a la etapa de procesamiento**

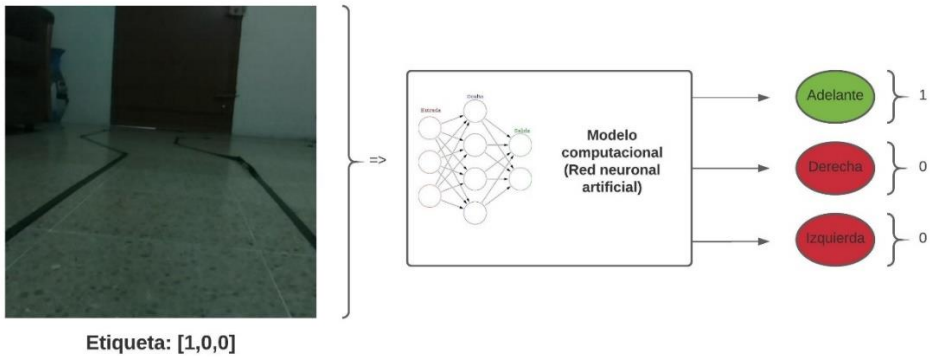


Fuente: elaboración propia, utilizando *Python* y *OpenCV*.

4.1.3. Etiquetado de imágenes

El etiquetado de imágenes consistió en asignar un código único numérico a la representación de una imagen, desde un punto de vista computacional se debe indicar a la máquina de tal forma que pueda comprender cuando se trata de un conjunto de categorías posible; a continuación en la figura 38 se muestra una representación gráfica del etiquetado de imágenes interpretado por un modelo computacional donde cada etiqueta representa una categoría específica, en nuestro caso el equivalente a distinguir entre: adelante, derecha e izquierda.

Figura 38. **Etiquetado de imágenes**



Fuente: elaboración propia, empleando la herramienta https://lucid.app/.

4.2. Creación del modelo computacional

Se presenta la definición e implementación de un modelo computacional utilizando redes neuronales que sea capaz de obtener como salida la dirección de movimiento del vehículo a escala a partir de una imagen capturada por la cámara en tiempo real.

4.2.1. La red neuronal propuesta

Se propone la utilización del perceptrón multicapa que se define como una red neuronal artificial formada por múltiples capas.

4.2.1.1. Capa de entrada

La capa de entrada está conformada por el vector de salida obtenido en la fase de procesamiento de la imagen, este consiste en un vector de una sola

dimensión con 57 600 unidades, cada una de estas representa un *pixel* en la imagen que originalmente tiene una dimensión de 240x240 *pixeles*.

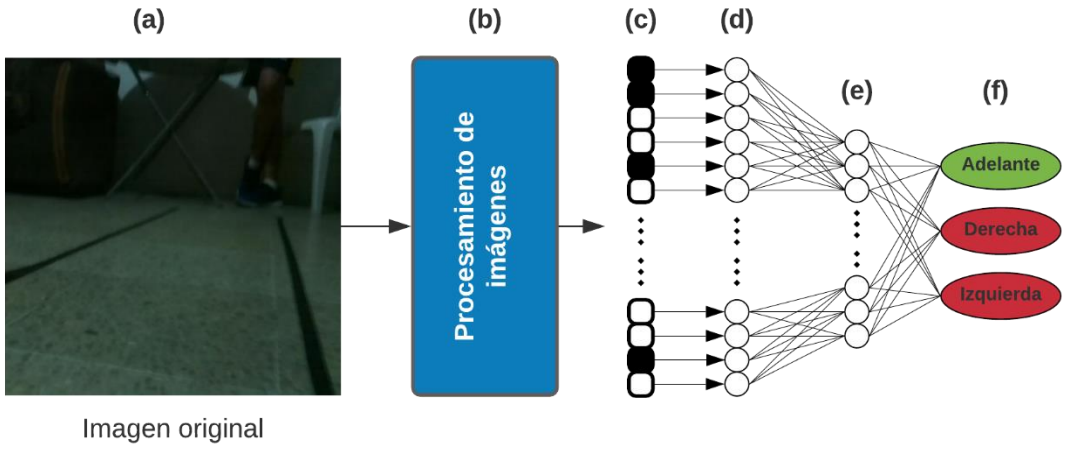
4.2.1.2. Capa de salida

La capa de salida está conformada por 3 unidades, cada una representa una dirección de movimiento distinta: adelante, derecha e izquierda.

4.2.1.3. Capas ocultas

Se definió una sola capa oculta con 32 unidades, sin embargo, es posible experimentar cambiando la cantidad de capas ocultas y las unidades de cada una. Se debe tomar en cuenta también que mientras más unidades y capas existan en nuestra red así será el esfuerzo computacional que se requerirá para poder encontrar los parámetros óptimos.

Figura 39. **Modelo computacional**



Fuente: elaboración propia, empleando la herramienta https://lucid.app/.

En la figura 39 se muestra un resumen gráfico del modelo computacional implementado, el cual se compone de los siguientes pasos: a) Obtención de los datos, b) y c) procesamiento de imágenes en el cual se obtiene como salida un arreglo unidimensional binario, d) la capa de entrada de la red neuronal artificial compuesta de 57 600 unidades, e) capa oculta de 32 unidades y f) capa de salida con 3 unidades que representan la dirección de movimiento a aproximar a partir de la imagen de entrada.

4.2.2. Entrenamiento de la red neuronal utilizando OpenCV

Definida la red neuronal, se deben ajustar los parámetros que interconectan cada una de las unidades, neuronas artificiales, que componen la red con el objetivo de crear un modelo computacional que pueda aproximar la dirección de movimiento a partir de una imagen como entrada y minimice el error en dicha aproximación.

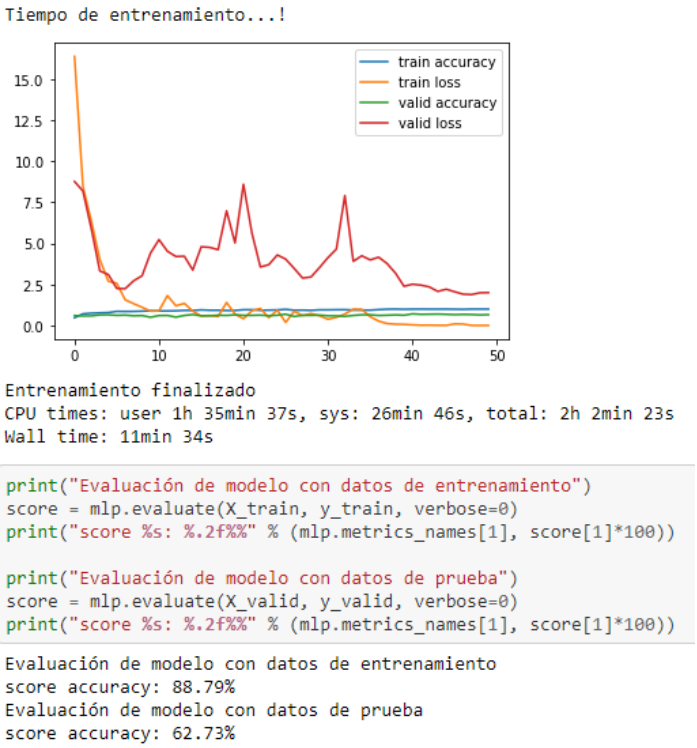
Para el entrenamiento de la red neuronal se debe definir una métrica de evaluación que nos indique como se desempeña la red de una forma numérica, para este trabajo se utilizó la exactitud del modelo que consiste en obtener el error a partir del valor inferido y el valor real; se aplicó el algoritmo descenso del gradiente para actualizar los valores de cada parámetro de la red iterando en el conjunto de datos completo, también se utilizó la técnica de retro propagación inversa que consiste en obtener el error a partir de cada una de las unidades de la red utilizando derivación compuesta y a partir de dicho valor se pueden actualizar los parámetros de la red con el objetivo de minimizar la función de coste o error.

Se debe definir un conjunto de datos para entrenamiento y uno distinto para la realización de pruebas, esto con el objetivo de evaluar el modelo y su

capacidad para generalizar con imágenes que nunca ha visto. El conjunto de datos utilizado para el entrenamiento de este modelo está definido de la siguiente manera:

- Entrenamiento, 80 %: 437 imágenes
- Pruebas, 20 %: 110 imágenes

Figura 40. **Resultado del entrenamiento de una red neuronal artificial**



Fuente: elaboración propia utilizando *Python* en un ambiente interactivo.

Como se observa en la figura 40 después del entrenamiento se obtuvo un modelo con exactitud del 88,79 % en el conjunto de datos de entrenamiento y el 62,73 % de exactitud para el conjunto de datos de prueba, en el eje **x** de la gráfica

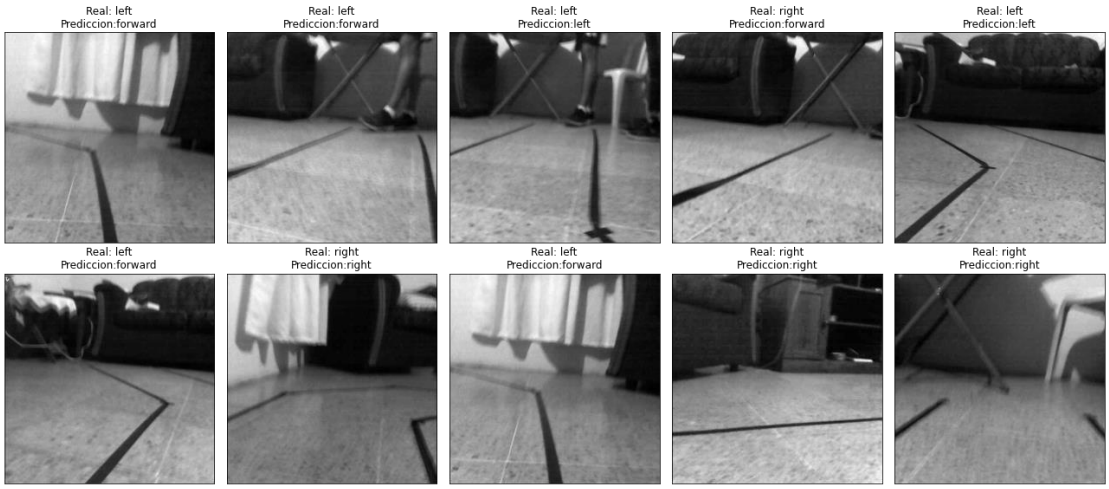
se muestran las iteraciones y en el eje **y** se muestra como cambio la exactitud del modelo en cada iteración.

4.3. Utilizando el modelo

Entrenado el modelo se exportan los parámetros obtenidos para cada capa de la red neuronal artificial en un archivo con una estructura matricial que depende del tamaño de cada una de las capas en la red, el modelo se exporto en un archivo con extensión *.h5* que contiene una estructura de datos numérica interpretable por la librería *tensorflow* de *Python* para su utilización en un ambiente real.

Se importo el modelo computacional creado para la realización de pruebas en un ambiente real, en la figura 41 se observa el resultado de aplicar el modelo a imágenes obtenidas por el prototipo creado; en cada imagen se muestra el valor de dirección esperado y el valor obtenido por el modelo de los cuales 5 de un total de 8 imágenes fueron etiquetadas de forma correcta.

Figura 41. **Utilización de modelo con datos reales**



Fuente: elaboración propia, empleando *Python* y *OpenCV*.

4.3.1. Configurar el prototipo para conducción autónoma

Para lograr que el prototipo se conduzca de forma autónoma se debe crear una función que reciba como entrada una imagen que debe actualizarse en periodos de tiempo cortos, aplicar las técnicas de procesamiento indicadas e ingresarlas al modelo computacional creado, la salida será un vector que indique la dirección de movimiento que debe tener el vehículo a escala en un momento dado.

CONCLUSIONES

1. Las técnicas de procesamiento de imágenes son importantes para obtener la mayor cantidad de información de una imagen en forma digital.

2. La creación de un modelo computacional utilizando redes neuronales artificiales para clasificar imágenes con dimensiones de 240px240p implica entrenar 57 600 parámetros y no es posible realizar el entrenamiento en un Raspberry Pi.

3. En un ordenador de placa reducida *Raspberry Pi* es posible aplicar técnicas de procesamiento de imágenes, utilizar modelos con redes neuronales artificiales para inferir la dirección del movimiento a partir de la captura de imágenes de una cámara en tiempo real.

4. Los modelos de redes neuronales artificiales no son la mejor opción para realizar clasificación de imágenes.

RECOMENDACIONES

1. Obtener el conjunto de datos en condiciones de luz distintas para que el modelo computacional sea robusto y pueda detectar dichos cambios.

2. Utilizar un *Raspberry Pi* de última generación, para tener mayor capacidad computacional.

3. Actualizarse constantemente, nunca dejar de aprender. El campo de la inteligencia artificial avanza aceleradamente y se van creando nuevos conocimientos que ayudan a resolver este tipo de problemas con mejores resultados por lo que debemos actualizar nuestro conocimiento para estar a la vanguardia.

4. Utilizar modelos de redes neuronales artificiales creados con el propósito de manejar imágenes, por ejemplo, las redes neuronales convolucionales.

BIBLIOGRAFÍA

1. GOODFELLOW, Ian; BENGIO, Yoshua. *Deeplearning Book.* [en línea]. <https://www.deeplearningbook.org/>. [Consulta: 20 de abril de 2020].

2. MARTÍNEZ, Virginia. *Introducción a la plataforma Arduino y al sensor ultrasónico HC-SR04: experimentando en una aplicación para medición de distancias.* [en línea]. <https://e-archivo.uc3m.es/bitstream/handle/10016/22916/TFG_Virginia_Martinez_Fuentes.pdf>. [Consulta 18 de febrero de 2020].

3. OpenCV. *Image processing in OpenCV.* [en línea] <https://docs.opencv.org/master/d2/d96/tutorial_py_table_of_contents_imgproc.html>. [Consulta: 10 de febrero de 2020].

4. _____. *Camera calibration.* [en línea] < https://docs.opencv.org/master/dc/dbb/tutorial_py_calibration.html> . [Consulta: 10 de agosto de 2020].

5. PARKER, Jim. *Algorithms for image processing and computer vision*. [en línea] <https://books.google.com.gt/books?hl=es&lr=&id=BK3oXzpxC44C&oi=fnd&pg=PR1&dq=image+processing&ots=IThGfgBGLI&sig=szITMEcUkXXPf6ACO8wStK20Z3k&redir_esc=y#v=onepage&q=image%20processing&f=false>. [Consulta: 15 de marzo de 2020].

6. Pyimagesearch. *Accessing the Raspberry Pi Camera with OpenCV and Python.* [en línea]. <https://www.pyimagesearch.com/2015/03/30/accessing-the-raspberry-pi-camera-with-opencv-and-python/>. [Consulta 16 de agosto de 2020].

7. Raspberry Pi. *Setting up Raspberry Pi.* [en línea]. <https://projects.raspberrypi.org/en/pathways/getting-started-with-raspberry-pi>. [Consulta: 5 de agosto de 2020].

8. UPTON, Eben. *Meet the Raspberry Pi.* [en línea] < https://books.google.com.gt/books?id=C0bZSmKnRK4C&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false>. [Consulta: 16 de febrero de 2020].

9. YEGNANARAYANA, Bayya. *Artificial neural networks*. [en línea] <https://books.google.com.gt/books?hl=es&lr=&id=RTtvUVU_xL4C&oi=fnd&pg=PR9&dq=artificial+neural+networks&ots=Gd6-BizERy&sig=-1SdTffFxIGyxGlLn2P6XUlN-6U&redir_esc=y#v=onepage&q=artificial%20neural%20networks&f=false>. [Consulta: 20 de diciembre de 2019].

APÉNDICES

Clase implementada en *Python* para manejar los *GPIO* del *Raspberry Pi* que controlan los motores del vehículo a escala utilizando el módulo L298.

Apéndice 1. **Control de motores vehículo a escala**

```
# -*- coding: utf-8 -*-
"""
Autor: Jose Fernando Perez
Descripcion: Clase para controlar un modulo L298 para control de motores,
se utilizan los pines 6, 13, 19 y 26. No se utiliza el control de velocidad con PWM.
"""
try:
    import RPi.GPIO as gpio
except RuntimeError:
    print("Error al importar RPi.GPIO, puede que necesite permisos de super usuario.")

class RCDriver():
    """
    Se define la clase motor_handler, que nos permitira el control de 2 motores DC utilizando
    el modulo L298D; se implementan los metodos de inicialización, movimiento y finalización.
    """
    def __init__(self):
        self.MOTORS_PINS = [6, 13, 19, 26]
        self.VEL_PIN = 5
        self.velocidad = None
        self.movements = {
            'DERECHA'   : (0,1,0,1),
            'IZQUIERDA' : (0,1,1,0),
            'ADELANTE'  : (0,1,0,0),
            'ATRAS'     : (1,0,0,0),
            'DEFAULT'   : (0,0,0,0)
        }

    def motors_init(self):
        gpio.setmode(gpio.BCM)
        gpio.setwarnings(False)
        gpio.setup(self.MOTORS_PINS, gpio.OUT)
        gpio.setup(self.VEL_PIN, gpio.OUT)
        self.velocidad = gpio.PWM(self.VEL_PIN, 100)
        self.velocidad.start(55)

    def move(self, direction='DEFAULT'):
        gpio.output(self.MOTORS_PINS, self.movements[direction])

    def __exit__(self):
        self.move()
        gpio.cleanup()
```

Fuente: elaboración propia, empleando editor de código *Python IDLE*.

Archivo de configuración con valores constantes utilizados para el proyecto, se define el alto y ancho de la imagen.

Apéndice 2. **Archivo de configuración**

```
#Archivo para manejar los valores constantes de configuracion
#Autor: Jose Fernando Perez

#PiCamera
HEIGHT = 480
WIDTH = 480
FPS = 30
```

Fuente: elaboración propia, empleando editor de código *Python IDLE*.

Clase implementada en *Python* para obtener imágenes del módulo de cámara y dirección de movimiento desde el *Raspberry Pi.*

Apéndice 3. **Clase en *Python* para obtener imágenes y dirección de movimiento**

```
class collect_data:
    def __init__(self, cam_HEIGHT, cam_WIDTH, cam_FPS):
        self.camera = picamera.PiCamera()
        self.camera.resolution = (cam_HEIGHT, cam_WIDTH)
        self.camera.framerate = cam_FPS
        print("Iniciando PiCamera...")
        time.sleep(2)
        print("PiCamera iniciada")
        self.car = RCDriver()
        self.car.motors_init()
        pygame.init()
        pygame.display.set_mode((250,250))

    def collect(self):
        clock = pygame.time.Clock()
        while True:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    pygame.quit()
                    break

            keys_pressed = pygame.key.get_pressed()
            direction = 'DEFAULT'
            if keys_pressed[pygame.K_LEFT]:
                self.car.move(direction='IZQUIERDA')
                direction = 'left'
            elif keys_pressed[pygame.K_RIGHT]:
                self.car.move(direction='DERECHA')
                direction = 'right'
            elif keys_pressed[pygame.K_UP]:
                self.car.move(direction='ADELANTE')
                direction = 'forward'
            elif keys_pressed[pygame.K_DOWN]:
                self.car.move(direction='ATRAS')
            elif keys_pressed[pygame.K_q]:
                break
            else:
                self.car.move()

            stream = io.BytesIO()
            self.camera.capture(stream, format='jpeg', use_video_port=True)
            self.save_image(stream, direction)
            stream.flush()
            clock.tick(30)
        pygame.quit()
        self.car.__exit__()

    def save_image(self, s, d):
        s.seek(0)
        if d in ['left', 'right','forward']:
            image = Image.open(s)
            stamp = time.ctime(time.time()).replace(" ", "_")
            image.save(f"/home/pi/Documents/Tesis/autonomous-rcar-pi/{d}/image_{stamp}", format="JPEG")
```

Fuente: elaboración propia, empleando editor de código *Python IDLE*.

Definición de red neuronal artificial utilizando la librería *OpenCV*,

Apéndice 4. **Red neuronal artificial en *OpenCV***

Definición clase para modelo de redes neuronales

```
class MLP:
    def __init__(self):
        self.model = None

    def create_mlp(self, layers_dim):
        self.model = cv2.ml.ANN_MLP_create()
        self.model.setLayerSizes(np.int32(layers_dim))
        self.model.setTrainMethod(cv2.ml.ANN_MLP_BACKPROP)
        self.model.setActivationFunction(cv2.ml.ANN_MLP_SIGMOID_SYM, 2, 1)
        self.model.setTermCriteria((cv2.TERM_CRITERIA_COUNT, 100, 0.01))

    def train(self, X, y):
        start = time.time()

        print("Entrenamiento en curso...")
        self.model.train(np.float32(X), cv2.ml.ROW_SAMPLE, np.float32(y))

        end = time.time()
        print("Entrenamiento finalizado en: %.2fs" % (end-start))

    def evaluate(self, X, y):
        ret, resp = self.model.predict(X)
        prediction = resp.argmax(-1)
        true_labels = y.argmax(-1)
        accuracy = np.mean(prediction == true_labels)
        return accuracy

    def save_model(self, name):
        path = "models/"+name
        self.model.save(path)
        print("El modelo se guardo en: {path}")


    def load_model(self, path):
        if not os.path.exists(path):
            print("El modelo no existe")
            sys.exit()
        self.model = cv2.ml.ANN_MLP_load(path)

    def predict(self, X):
        resp = None
        try:
            ret, resp = self.model.predict(X)
        except Exception as e:
            print(e)
        return resp.argmax(-1)
```

Fuente: elaboración propia, empleando editor de código *Python IDLE*.

Función implementada en *Python* para procesamiento de imágenes, redimensionar el tamaño a 240x240*pixeles*, convertir a escala de grises, aplicación de filtro *gaussiano* para eliminación de ruido, detección de bordes y creación de arreglo unidimensional.

Apéndice 5. **Procesamiento de imágenes**

```python
def image_preprocessing(img):
    # Redimensionamiento de la imagen
    img = cv2.resize(img, (WIDTH, HEIGHT))
    # Conversion a escala de grises
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Aplicacion de filtro
    gray = cv2.medianBlur(gray, 5)
    # Obtener bordes
    canny = cv2.Canny(gray, 50, 75)
    # Convertir a arreglo unidimensional
    X = canny.reshape(1, -1)
    return X
```

Fuente: elaboración propia, empleando editor de código *Python IDLE*.

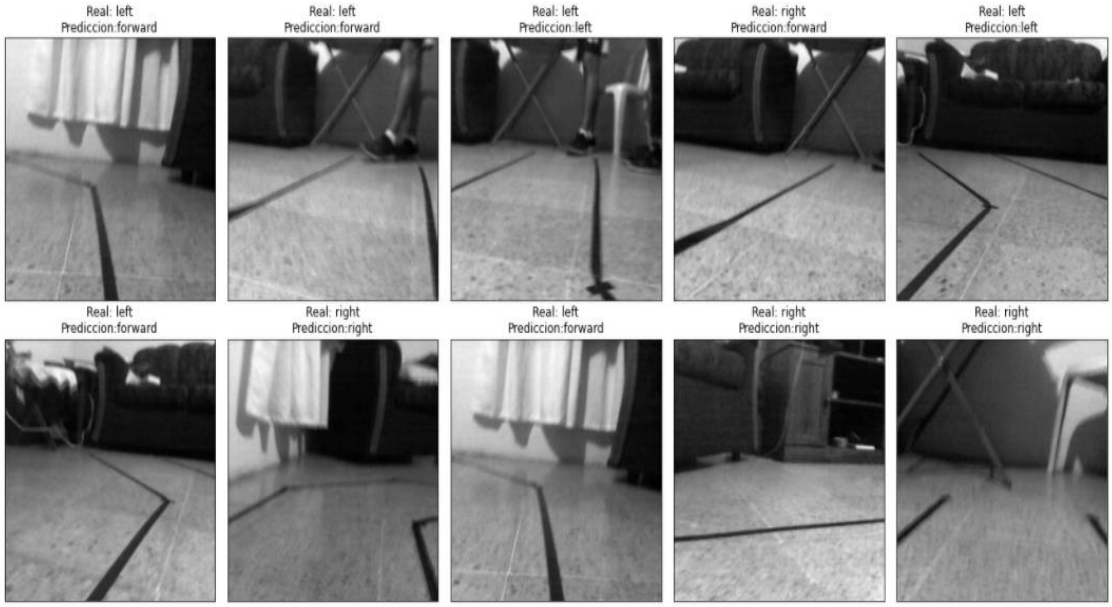
Función implementada en *Python* para utilización del modelo computacional y mostrar resultados.

Apéndice 6. **Resultado final**

```
rand_cat = [random.randrange(0, 3, 1) for i in range(10)]
rand_img = [random.randrange(0, 100, 1) for i in range(10)]
images = []
for i, z in zip(rand_cat, rand_img):
    images.append(f"{DATA_PATH}{CATEGORIES[i]}/{z}.jpeg")
#print(images)
fig, axs = plt.subplots(nrows=2, ncols=5,
                        figsize=(18,8), subplot_kw={'xticks': [], 'yticks': []})

for ax, img_path in zip(axs.flat, images):
    img = cv2.imread(img_path)
    X = image_preprocessing(img)
    predicted = np.argmax(model.predict(X), axis=-1)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img = cv2.resize(img, (WIDTH, HEIGHT))
    ax.imshow(img, cmap='gray')
    title = f'Real: {str(img_path.split("/")[1])}\nPrediccion:{CATEGORIES[predicted.item(0)]}'
    ax.set_title(title)

plt.tight_layout()
plt.show()
```



Fuente: elaboración propia, empleando editor de código *Python IDLE*.