# Aplicación de Algoritmos de Deep Learning en un Sistema Embebido para el control de una Mano Robótica


Juan Camilo Cuervo Restrepo

Harlinsson Javier Chavarro Hurtado

Ingeniería Mecatrónica y Electrónica

Dr. Carlos Andrés Madrigal

Dr. Héctor Erives



**INSTITUTO TECNOLÓGICO METROPOLITANO**

**27 de julio del 2018**

# RESUMEN

Los sistemas embebidos son muy útiles para aplicaciones, ya que son compactos y sencillos de aprender a programar, pero a pesar de estas características tiene un problema, el cual son muy limitados debido a su baja capacidad de procesamiento para llevar a cabo tareas de Deep Learning. Otros de los problemas es el comportamiento de los módulos de PWM de las Raspberry no es óptimo, debido a que no es capaz de mantener un periodo constante, por ende, cada uno de los actuadores mantiene un pequeño movimiento mientras están energizados.

El presente trabajo de investigación se fundamentó en implementar una arquitectura de redes neuronales profundas - Deep Learning (DNN, por sus siglas en inglés) sobre el sistema embebido Raspberry pi, el cual se encarga de controlar una mano robótica. ¿Pero cómo lo hace? Por medio de una cámara se adquiere una imagen del objeto a detectar, y el DNN se encarga de reconocer el objeto y entregar el rectángulo delimitante sobre la imagen. Luego se procede a ordenar la activación de los mini servos para hacer cerrar los dedos de la mano robótica de una manera precisa para lograr agarrar el objeto adecuadamente.

Para un mejor rendimiento del algoritmo Deep Learning en la Raspberry se utiliza un modelo pre entrenado con pocas clases, es decir, un modelo que es capaz de reconocer pocos objetos. La programación se realizó en Python, apoyándonos con librerías como OpenCV y Caffe, ya que estas fueron las que dieron mejores resultados sobre la Raspberry pi. Se probaron otros modelos de librerías diferentes, pero requerían de mucho procesamiento o simplemente no funcionaban sobre el sistema operativo Raspbian de la Rasberry Pi.

Para el diseño de la mano robótica nos apoyamos en un modelo ya existente, al cual se le realizo modificaciones para una mejor movilidad, se redimensionaron piezas y se añadieron nuevos modelos, también se cambiaron algunos materiales para dar mayor rendimiento al movimiento de la mano, el diseño del color fue escogido de tal manera que resaltara a la vista del público.

Actualmente hay muchas personas apostándole al Deep Learning sobre dispositivos embebidos, ya que estos dispositivos dependiendo de las aplicaciones son mucho más flexibles al implementar determinadas tareas debido a su tamaño reducido y bajo costo.

***Palabras clave*:** Raspberry Pi, Deep Learning, Python, Mano robótica

# RECONOCIMIENTOS

En primer lugar, agradecer a Dios, por habernos permitido estar en esta experiencia, por darnos tiempo, paciencia e inteligencia en el desarrollo del presente trabajo de grado. Agradecemos enormemente a nuestros docentes asesores que estuvieron atentos en el apoyo de la investigación al Dr. Carlos Andrés Madrigal, Dr. Héctor Erives, al Dr. Sergio Cabrera a los diferentes profesores por el conocimiento compartido, a las dos universidades tanto UTEP como el ITM por la oportunidad que se brindó para el desarrollo de este proyecto, y al personal de los laboratorios de la UTEP, los cuales nos brindaron su apoyo para facilitar herramientas necesarias para llevar con éxito esta investigación, agradecimiento a Colciencias y a 100.000 Strong in the Americas por su apoyo.

Agradecer a la Dra. María Constanza Torres, Dra. Luisa Arvizu y al Dr. Miguel Vélez por su gran compromiso y responsabilidad con nosotros.

Agradecer al Héctor E. Lugo y a Ralph Loya por sus servicios y sus contribuciones en el proyecto.

También a nuestras familias que estuvieron presentes todo el tiempo y nos brindaron su apoyo moral a lo largo de la realización de nuestro trabajo.

# ACRÓNIMOS

***PWM*** *Pulse Width Modulation*. Modulación por ancho de pulsos

***GPU*** *Graphics Processing Unit.* Unidad gráfica de proceso.

***CPU*** *Central Process Unity*. Unidad central de proceso.

***IA*** *Artificial Intelligence*. Inteligencia Artificial

***API*** Application Programming Interface. Interfaz de programación de aplicaciones

***CAD*** Computer Assistant Design. Diseño Asistido por Computadora

***DNN*** Deep Learning neural network. Redes Neuronales Profundas

# TABLA DE CONTENIDO

# 1. INTRODUCCIÓN

**GENERALIDADES**

El presente siglo es muy popular por ser el auge de la tecnología, es el momento donde se ha diversificado y expandido de manera exponencial y a una gran velocidad. En Colombia también nos hemos adaptado a estos cambios tecnológicos. En la actualidad investigadores y científicos apuesta a la inteligencia artificial para realizar diferentes tareas, desde detección, clasificación, reconocimiento y predicción en diferentes campos de la ingeniería, esto con el fin de mejorar los procesos industriales y cotidianos. Para este caso se utiliza Deep Learning para la detección de objetos básicos del hogar, usando un modelo pre-entrenado aplicándolo a un sistema embebido.

**OBJETIVO GENERAL**

- Implementar una arquitectura de Deep Learning sobre un sistema embebido para realizar el control de una mano robótica encargado de tomar objetos específicos.

**OBJETIVOS ESPECÍFICOS**

- Determinar el esquema de control con las características del sistema embebido
- Desarrollar el control de los actuadores y sensores que se involucran en los movimientos del brazo robótico
- Implementación de un modelo pre-entrenado para reconocer objetos

**ORGANIZACIÓN DE LA TESIS**

En este proyecto está dividido en 3 parte principales, la primera parte consta de la construcción de la mano, segundo la etapa de reconocimiento (Deep Learning) y por último el control de los mini servos para el movimiento de la mano robótica.

En la sección de marco teórico se presentan las definiciones y cada una de las herramientas utilizadas para el desarrollo de este proyecto. En la metodología se expone como se realizó el trabajo con cada uno de los pasos para lograr obtener los resultados.

# 2. MARCO TEÓRICO

Actualmente hay muchos algoritmos desarrollados para tareas específicas, algoritmos para clasificar productos o reconocer patrones. A partir de esto se emplea un conjunto de algoritmos llamados aprendizaje automático o también conocidos como Machine Learning, el cual es una rama de la inteligencia artificial que utiliza técnicas para que las computadoras aprendan. En nuestro caso se utiliza el Deep-Learning (Aprendizaje Profundo) el cual es un conjunto de algoritmo de la clase de Machine Learning, también pertenece al campo de la inteligencia artificial.

**DEEP LEARNING**

En español Aprendizaje Profundo, es un campo que pertenece a la inteligencia artificial, el cual tiene como objetivo el de imitar el aprendizaje de los seres humanos con algoritmos extensos para llegar a ser capaces de aprender a partir de las experiencia y ejemplos, la red neuronal profunda está compuesta por un grupo de neuronas donde cada neurona adquiere unos “pesos” de conexión, de acuerdo a la base de datos de entrenamiento, mediante la interconexión entre más neuronas la capacita para reconocer, clasificar y detectar información. Dicho de otra manera, el Deep Learning usa redes neuronales, el cual es un sistema de programas y estructuras que se parece mucho al funcionamiento del cerebro humano. Estas redes neuronales como su nombre lo indica, hay varias neuronas que están interconectadas entre sí, cada una con sus datos y al estar todas en una conexión, son capaces de aprender. Actualmente grandes empresas como Google, Tesla, Facebook usan Deep Learning porque saben del potencial que tienen. El año pasado Facebook creó la unidad de investigación en IA, con el fin de ayudar a crear soluciones para identificar mejor rostros y objetos que aparecen en los 350 millones de videos y fotografías que se transfieren diariamente a Facebook. Un ejemplo de Deep Learning es el sistema de Apple denominado Siri (Banafa, 2016).

Para el futuro el Deep Learning garantiza muchos avances, por ejemplo, la conducción de vehículos autónomos donde están participando empresas como Google y Tesla, entre otras (Banafa, 2016).

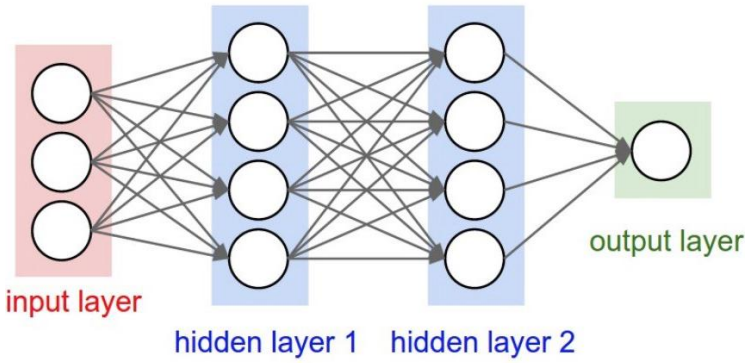


*Figura 1. Arquitectura Deep Learning con 3 entradas, 2 capas de neuronas ocultas y una salida.*

En la figura 1. Se puede apreciar una arquitectura de Deep Learning con dos capaz ocultas, tres entrada y una salida.

El Deep Learning en este proyecto aporta la parte de detección de objetos, donde se usa un modelo pre entrenado capaz de reconocer diversos objetos, entre estos, las botellas, la tomaremos de ejemplo para explicar el funcionamiento de nuestro proyecto. El Deep Learning necesita mucho procesamiento en una maquina computacional, por eso es recomendado utilizar GPU unidad de procesamiento gráfico, ayuda al CPU a agilizar el trabajo sobre todo a la hora de correr programas o algoritmos pesados computacionalmente. Debido a que se corrieron las DNN sobre el sistema embebido Raspberry Pi, se optó por utilizar Deep Learning con OpenCV.

**OpenCV**

Sus siglas en ingles significan Open Source Computer Vision Library, en español librería de visión de computador de código abierto. Puede ser programada en interfaces como C++, Python y Java, además es compatible con todos los sistemas operativos, fue diseñado para

buscar la eficiencia computacional con un gran enfoque en aplicaciones de tiempo real. Esta librería es adoptada en todo el mundo con 47 mil usuarios. Con esta librería es mucho más fácil leer imágenes para su procesamiento, ya que tiene funciones que nos facilita ya sea para tomar una foto, hacer video, aplicar diferentes filtros, entre otras funcionalidades. (team, 2018)

**Caffe**

Este Framework para el desarrollo de DNNs se enfoca especialmente en la visión artificial que es lo que nos interesa a nosotros y es muy bueno haciéndolo, además es muy robusto y rápido a comparación de otros framework que son mucho más pesados computacionalmente y requieren de más procesamiento. Por estas razones nombradas se trabajó con un modelo de la librería Caffe, además las clases con la que fue entrenada la arquitectura de Deep Learning son suficientes para la implementación sobre el sistema embebido.

**Python**

Es uno de los lenguajes de programación más sencillos que existen en el momento, es un lenguaje interpretado y multiplataformas. Es un lenguaje de código abierto, tiene demasiada información para aprender a programar en él. Hay muchas comunidades que organizan conferencias para brindar ayuda sobre este lenguaje, al tener muchas librerías hace más fácil su implementación para el usuario. Por estas razones se lleva a cabo el proyecto en este lenguaje, ya que es soportado por el sistema operativo de la Raspberry Pi Raspbian.

**RASPBERRY PI 2**

La Raspberry pi es una computadora de tarjeta reducida, el software que utiliza esta tarjeta es libre (Open Source) y su sistema operativo es de la distribución de Debian (Linux) llamada Rasbian. Fue lanzada en el año 2014 utiliza un modelo de procesador llamado BCM2836,

con cuatro núcleos, tiene una memoria RAM de 1GB nada mal para un sistema embebido, incluye 40 pines GPIO con 4 puertos USB para conectar diferentes periféricos. Tiene una CPU de 900MHz quad-core ARM Cortex A7. Con estas especificaciones que tiene esta tarjeta electrónica, fueron suficientes para implementar el código de Deep Learning.

Algunas aplicaciones sobre sistemas embebidos con algoritmos de Deep Learning o usando cada uno de sus periféricos descritos a continuación:

(Sajjad, 2017) Aplican una metodología para el reconocimiento facial en sistemas embebidos para realizar una detección de rostros de manera portable. Este articulo está dirigido para reconocimiento facial utilizado por policías para la ayuda de identificar de manera más rápida a las personas en tiempo real, con una cámara inalámbrica pequeña en el uniforme del oficial para capturar así la información del video que va a estar pasando por la Raspberry pi para la respectiva detección y reconocimiento de cada persona. El método propuesto se implementa en una Raspberry Pi 3 Model B en Python 2.7.

(Amato, 2016) Utiliza un sistema embebido para calcular el número de espacios libres en un parqueadero de automóviles implementado visión artificial.

(PeterO'Donovan, 2018) Hace una Implementación de dispositivos como Raspberry Pi y demás sistemas embebido apoyándose de la industria 4.0 y usando la nube como base de datos para realizar los cálculos necesarios.

(Anandhalli, 2017) Realiza tracking de vehículos en tiempo real mediante el sistema embebido Raspberry Pi, en el cual utilizan video RGB para convertirlo después en diferentes espacios de colores y así eliminar cualquier tipo de ruido que pueda ocasionar problema con el tracking de los automóviles.

(Jidong, 2015) Propone el uso de un sistema embebido para reconocer frutas en su ambiente natural, implementando técnicas de visión artificial, tales como filtros para determinar bordes y segmentar el tipo de fruta.

(Karl-HeinzHerrmann, 2014) Intenta implementar diseños realizados con impresoras 3D para replicar partes del cuerpo humano, donde las piezas realizadas sean compatibles y no exista la posibilidad de rechazo por parte del cuerpo.

Por otro lado, hablando del auge y constante desarrollo en sistemas embebidos contamos con las siguientes investigaciones.

(Alippi, 2016) En dicho artículo se resalta la importancia de los sistemas embebidos en la industria actual, debido a sus grandes ventajas en términos de rendimiento comparado con su tamaño, así como la infinidad de aplicaciones realizadas en estos dispositivos.

(John, 2015) Propone la aplicación de algoritmos matemáticos sobre la Raspberry Pi para la clasificación de multi Etiquetas, aprovechando el bajo costo del sistema embebido para realizar cálculos complejos sobre arquitecturas sencillas.

(Redmon, 2018) Aplican el modelo YOLO (You Only Look Once), en español Tu solo lo miras una sola vez, es un detector muy rápido y preciso comparados con otros. El sistema de detección previa reutiliza clasificadores para realizar la detección. Aquí se aplica a una red neuronal única a la imagen completa, la red neuronal divide la imagen en regiones y predice cuadros de limite y probabilidades para cada región.

(Jing, 2018) Se propone el uso de la herramienta YOLO para el análisis de video y la segmentación del mismo para la extracción de características para el posterior estudio de cada una de ellas por separado.

(Pathak, 2018) Resaltan la importancia del uso de algoritmos de Deep Learning para la detección de diferentes objetos, todo esto apoyándose en la gran ayuda que brinda la visión artificial como parte fundamental de esta práctica. Además de esto se realizan distintos usos de las redes neuronales convoluciones para la detección, clasificación, localización y reconocimiento de distintas clases.

# 3. METODOLOGÍA

Este brazo fue construido de un material llamado PLA, donde sus siglas significa Acido Poli láctico, este material es creado a partir de recursos naturales, como la caña de azúcar, almidón de maíz entre otras. Debido a que este material es biodegradable es muy utilizado para impresiones 3D, ya que no produce gases contaminantes. El brazo robótico tiene acceso a los movimientos de los dedos y una rotación en la parte del codo, se realizó de una manera sencilla para que lograra tomar objetos pequeños, como una botella.

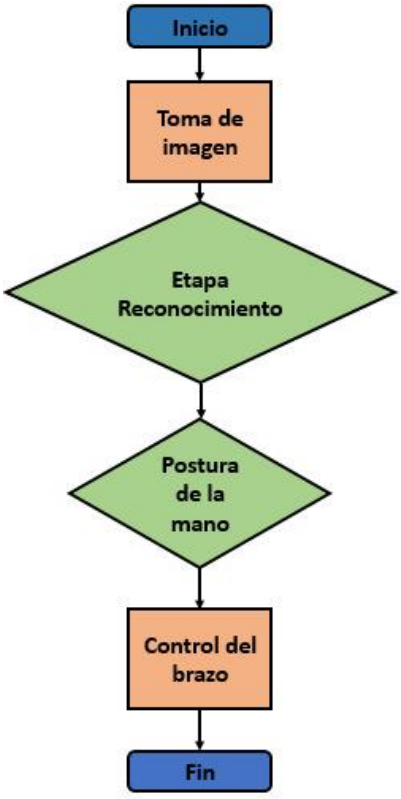


*Figura 2. Diagrama de Flujo del comportamiento del sistema*

En la figura anterior se observa un esquema metodológico desarrollado en este proyecto, donde se muestran los pasos que se realizaron para llevarlo a cabo, a continuación se explica los pasos.

*Figura 3. Captura de la imagen*

**Toma de Imagen**

Esta primera etapa del proceso empieza cuando se hace la toma de la fotografía para proceder con el reconocimiento, como se observa en la figura 3. dicha toma se realiza con la importación de la librería getch que permite conocer la captura de una tecla del teclado para entrar a un ciclo que guarde la imagen con un nombre específico para luego servir de entrada en el algoritmo de Deep Learning.

**Etapa de Reconocimiento**

Una vez se tenga la imagen se lleva a la API, encargada de realizar la ejecución del algoritmo y deducir a qué clase específica pertenecer uno o más objetos captados por la cámara.

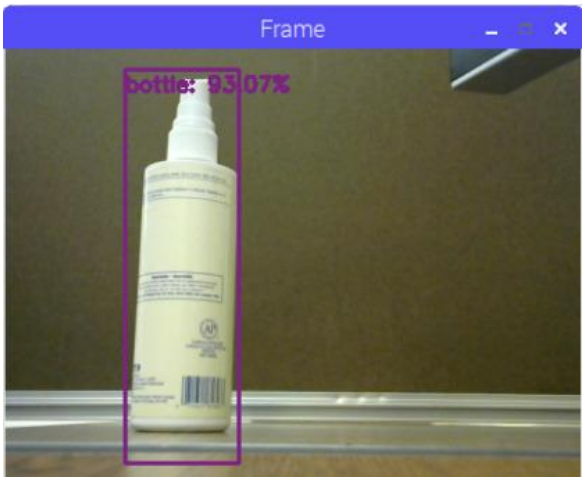


*Figura 4. Botella detectada por el Deep Learning*

Cuando se realiza la detección, en la imagen se encierra en pequeños cuadros delimitantes de colores a cada uno de los objetos que han sido reconocidos, ubicando su nombre, así como el porcentaje de precisión al lado de cada uno de ellos como se muestra en la imagen de la figura 4, además por el terminal se muestra un texto cuando es detectada la imagen también como se muestra en la siguiente figura 5.

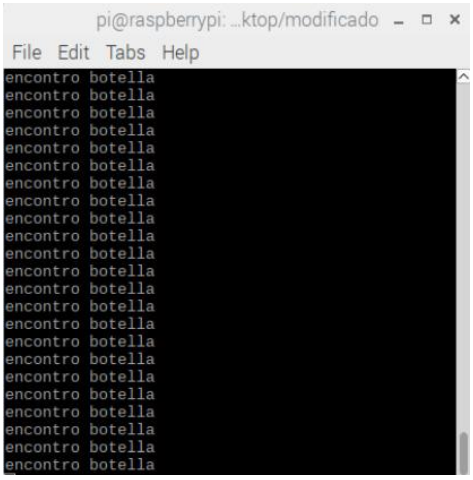


*Figura 5. Imagen donde se muestra detección de la botella*

**Postura de la Mano**

Tenemos la posición inicial de la mano en otras palabras la postura de la mano, que finalmente luego de detectar el objeto, se procede al control de la mano para tomar el objeto.

**Control de la Mano**

se han creado funciones para saber qué forma debe tomar la mano para poder agarrar el objeto adecuadamente, esto se hace comparando el nombre de la clase detectada entre diferentes funciones que contienen cada uno de los movimientos de los servomotores, establecidos con anterioridad. Por ejemplo, la postura de cada uno de los dedos para una botella fue establecida antes y se conoce los grados exactos que debe recorrer cada actuador para la mano pueda hacer su función de agarrarla. A continuación, se muestra una imagen donde se ven los mini servos que controlan el movimiento de la mano.

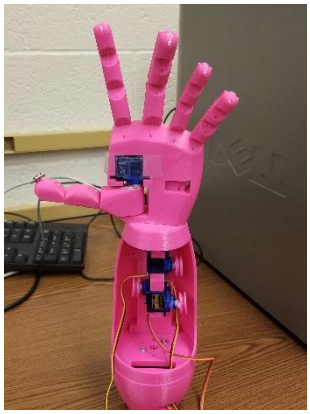

*Figura 6. Mano Robótica*

Finalmente, al tomar el objeto con el grado de libertad del codo de la mano podemos darle un giro y luego poner el objeto en otra parte para clasificarlo.

Muchas personas han tratado de trabajar con Deep Learning sobre la Raspberry Pi, pero han encontrado que esta tarjeta electrónica tiene recursos muy limitados, por esta razón algunos programadores tratan de aplicar modelos sencillos y utilizar diferentes tipos de

arquitecturas de Deep Learning para mejorar desempeño tanto en la precisión como en la velocidad de la detección.

# 4. RESULTADOS Y DISCUSIÓN

A pesar de las limitaciones de la Raspberry Pi con su procesamiento y poca memoria RAM se logra implementar el algoritmo de Deep Learning, realizando con éxito la detección de objetos en un tiempo promedio de 3 segundos una vez se hace la toma de la foto, así como el buen control por parte de los motores hacia la mano robótica a pesar del impreciso control del Dutty Cycle por parte de los módulos PWM ubicados en la Raspberry Pi.

Durante la realización de la mano robótica por parte de las impresoras 3D se obtuvieron varias versiones de esta, analizando cada caso en términos de movilidad y resistencia de los materiales de acuerdo con los tipos de construcciones en el CAD, se escogió una versión donde las articulaciones contaba de material elástico, que permitió volver a la posición inicial a cada uno de los dedos una vez hayan sido flexionados por parte de los mini servomotores.

Hablando en términos del desempeño del algoritmo de reconocimiento, los resultados fueron satisfactorios debido a la cantidad de clases que contenía el modelo pre entrenado, ya que a menor cantidad de clases tenga un modelo, más rápida será su lectura para reconocer objetos. El modelo cuenta con veintiuna clases para ser reconocidas, dentro de estos se encuentran sofás, botellas, con los cuales se obtuvieron resultados de hasta un 98% en términos de exactitud.

*Tabla 1. Comparación entre los distintos Framework*

| **Comparación de Framework** | | |
|---|---|---|
| **Frameworks** | **Tiempo de ejecución** | **Precisión** |
| Tensor Flow | 10 segundos | 92% |
| Keras | Capacidad Insuficiente para su ejecución | Capacidad Insuficiente para su ejecución |
| Caffe | 3 segundos | 95% |

Según la tabla nos damos cuenta de que el mejor modelo para usar en la Raspberry Pi es Caffe, debido a su buena velocidad y precisión al realizar el reconocimiento de un objeto.

El framework Keras al ejecutarse en la Raspberry Pi, era tan pesado y requería demasiado procesamiento, por esa razón cuando se ejecutó en la Raspberry, esta no respondía y se bloqueaba, además esta librería para su funcionamiento tiene como requisito primero instalar tensor Flow y sabemos que tensor Flow también es muy pesada para este dispositivo.

# 5. CONCLUSIONES, RECOMENDACIONES Y TRABAJO FUTURO

- El uso de algoritmos de Deep Learning en sistemas embebidos permite realizar tareas complejas a bajo costo y con excelentes prestaciones.

- Al aplicar algoritmos inteligentes en dispositivos como Raspberry es necesario ajustar el modelo pre-entrenado para las clases específicas a tratar.

- Utilizando herramientas de prototipado rápido como impresoras 3D, se pueden elaborar componentes de manera sencilla, a bajo costo y con buenas prestaciones en términos de resistencia de material.

Se recomienda utilizar la Raspberry Pi 3 Model B, ya que tiene mejores características para ejecución de algoritmos de Deep Learning, además esta tarjeta es mejorada en su rendimiento.

Para trabajos futuros se plantea entrenar nuestro propio modelo con clases específicas, para mejorar el rendimiento de la Raspberry Pi. También inclinarnos en la investigación de otras librerías de visión artificial como Theano, Pytorch, CNTK, MXNet para evaluar su velocidad y rendimiento.

# REFERENCIAS

Alippi, P. (2016). *A unique timely moment for embedding intelligence in applications.* Italiana, Switzerland.

Amato, G. (2016). *Deep learning for decentralized parking lot occupancy detection.* Pisa.

Anandhalli, M. (2017). *A novel approach in real-time vehicle detection and tracking using Raspberry Pi.* Hubballi, India .

Banafa, A. (7 de 08 de 2016). *OpenMind*. Obtenido de https://www.bbvaopenmind.com/que-es-el-aprendijaze-profundo/

Jidong, L. (2015). *Recognition of apple fruit in natural environment.* Zhenjiang, China.

Jing, L. (2018). *Video you only look once: Overall temporal convolutions for action recognition.* New York, United State.

John, N. (2015). *A Low Cost Implementation of Multi-label Classification Algorithm Using Mathematica on Raspberry Pi☆.* Tamilnadu, India.

Karl-HeinzHerrmann. (2014). *3D printing of MRI compatible components: Why every MRI research group should have a low-budget 3D printer.* Jena, Alemania.

Pathak, A. R. (2018). *Application of Deep Learning for Object Detection.* Bhubaneswar, India.

PeterO'Donovan. (2018). *A fog computing industrial cyber-physical system for embedded low-latency machine learning Industry 4.0 applications.* Cork, Irland.

Redmon, J. (2018). *You Only Look Once: Unified, Real-Time Object Detection.* Estados Unidos.

team, O. (2018). *OpenCV*. Obtenido de https://opencv.org/

# APÉNDICE

Código completo y reporte de actividades realizadas durante la investigación.

```
                  # import the necessary packages
from imutils.video import VideoStream
from imutils.video import FPS
from multiprocessing import Process
from multiprocessing import Queue
import numpy as np
import argparse
import imutils
import time
import cv2
import getch
import time
import RPi.GPIO as GPIO


########################## Pines PWM
#########################################
GPIO.setmode(GPIO.BOARD)
servoPin1=35 #Pulgar.
servoPin2=33 #Doblar pulgar e indice.
servoPin3=12 #Dedo Medio.
servoPin4=36 #Anular y Menhique

GPIO.setup(servoPin1, GPIO.OUT)

GPIO.setup(servoPin2,GPIO.OUT)
GPIO.setup(servoPin3,GPIO.OUT)
GPIO.setup(servoPin4,GPIO.OUT)

GPIO.setwarnings(False)

pwm1 =GPIO.PWM(servoPin1,50)
pwm2 =GPIO.PWM(servoPin2,50)
pwm3 =GPIO.PWM(servoPin3,50)
pwm4 =GPIO.PWM(servoPin4,50)
#############################################################
pwm1.start(0)
pwm2.start(0)
pwm3.start(0)
pwm4.start(0)
############# Functions Definition #################
def ceros():
    pwm1.ChangeDutyCycle(2.83)
```

```
    pwm2.ChangeDutyCycle(2.83)
    pwm3.ChangeDutyCycle(2.83)
    pwm4.ChangeDutyCycle(2.83)
def botella():
    DC1=1./18.*(60)+2
    pwm1.ChangeDutyCycle(DC1)
    ##
    DC2=1./18.*(150)+2
    pwm2.ChangeDutyCycle(DC2)
    ##
    DC3=1./18.*(150)+2
    pwm3.ChangeDutyCycle(DC3)
    ##
    DC4=1./18.*(160)+2
    pwm4.ChangeDutyCycle(DC4)
def persona():
    ceros()
    DC1=1./18.*(0)+2
    pwm1.ChangeDutyCycle(DC1)

    DC2=1./18.*(160)+2
    pwm2.ChangeDutyCycle(DC2)

    DC3=1./18.*(160)+2
    pwm3.ChangeDutyCycle(DC3)

    DC4=1./18.*(160)+2
    pwm4.ChangeDutyCycle(DC4)
    time.sleep(1.0)
    pwm1.ChangeDutyCycle(2.83)
    pwm2.ChangeDutyCycle(2.83)
    pwm3.ChangeDutyCycle(2.83)
    pwm4.ChangeDutyCycle(2.83)
    time.sleep(1.0)

    DC1=1./18.*(0)+2
    pwm1.ChangeDutyCycle(DC1)

    DC2=1./18.*(160)+2
    pwm2.ChangeDutyCycle(DC2)

    DC3=1./18.*(160)+2
    pwm3.ChangeDutyCycle(DC3)

    DC4=1./18.*(160)+2
    pwm4.ChangeDutyCycle(DC4)
    time.sleep(1.0)
    pwm1.ChangeDutyCycle(2.83)
    pwm2.ChangeDutyCycle(2.83)
    pwm3.ChangeDutyCycle(2.83)
```

```
    pwm4.ChangeDutyCycle(2.83)


################### Take a picture ###################
print('Press "a" to capture')
char=getch.getch()


if char=="a":
   camera=cv2.VideoCapture(0)
   return_value, vs=camera.read()
   cv2.imwrite('bot'+str(1)+'.jpg', vs)
   del(camera)


ceros()
def classify_frame(net, inputQueue, outputQueue):
     # keep looping
     while True:
          # check to see if there is a frame in our input queue
          if not inputQueue.empty():
               # grab the frame from the input queue, resize it,
and
               # construct a blob from it

               frame = inputQueue.get()
               frame=cv2.imread("bot1.jpg")
               frame = cv2.resize(frame, (300, 300))
               blob = cv2.dnn.blobFromImage(frame, 0.007843,
                    (300, 300), 127.5)

               # set the blob as input to our deep learning
object
               # detector and obtain the detections
               net.setInput(blob)
               detections = net.forward()

               # write the detections to the output queue
               outputQueue.put(detections)

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-p", "--prototxt", required=True,
     help="path to Caffe 'deploy' prototxt file")
ap.add_argument("-m", "--model", required=True,
     help="path to Caffe pre-trained model")
ap.add_argument("-c", "--confidence", type=float, default=0.2,
     help="minimum probability to filter weak detections")
args = vars(ap.parse_args())
```

```
# initialize the list of class labels MobileNet SSD was trained to
# detect, then generate a set of bounding box colors for each
class
CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",
      "bottle", "bus", "car", "cat", "chair", "cow", "diningtable",
      "dog", "horse", "motorbike", "person", "pottedplant",
"sheep",
      "sofa", "train", "tvmonitor"]
COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))

# load our serialized model from disk
print("[INFO] loading model...")
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])

# initialize the input queue (frames), output queue (detections),
# and the list of actual detections returned by the child process
inputQueue = Queue(maxsize=1)
outputQueue = Queue(maxsize=1)
detections = None

# construct a child process *indepedent* from our main process of
# execution
print("[INFO] starting process...")
p = Process(target=classify_frame, args=(net, inputQueue,
      outputQueue,))
p.daemon = True
p.start()

# initialize the video stream, allow the cammera sensor to warmup,
# and initialize the FPS counter
print("[INFO] starting video stream...")
frame=cv2.imread("bot1.jpg")
# vs = VideoStream(usePiCamera=True).start()
time.sleep(2.0)
fps = FPS().start()

# loop over the frames from the video stream
while True:
      # grab the frame from the threaded video stream, resize it,
and
      # grab its imensions
      frame=cv2.imread("bot1.jpg")
      frame = imutils.resize(frame, width=400)
      (fH, fW) = frame.shape[:2]

      # if the input queue *is* empty, give the current frame to
      # classify
      if inputQueue.empty():
            inputQueue.put(frame)
      # if the output queue *is not* empty, grab the detections
```

```
     if not outputQueue.empty():
          detections = outputQueue.get()
               print("entro")
     # check to see if our detectios are not None (and if so,
we'll
     # draw the detections on the frame)
     if detections is not None:
          # loop over the detections
          for i in np.arange(0, detections.shape[2]):
               # extract the confidence (i.e., probability)
associated
               # with the prediction
               confidence = detections[0, 0, i, 2]

               # filter out weak detections by ensuring the
`confidence`
               # is greater than the minimum confidence
               if confidence < args["confidence"]:
                    continue

               # otherwise, extract the index of the class label
from
               # the `detections`, then compute the (x, y)-
coordinates
               # of the bounding box for the object
               idx = int(detections[0, 0, i, 1])
               dims = np.array([fW, fH, fW, fH])
               box = detections[0, 0, i, 3:7] * dims
               (startX, startY, endX, endY) = box.astype("int")

               # draw the prediction on the frame

               label = "{}: {:.2f}%".format(CLASSES[idx],
                    confidence * 100)
               if confidence>0.60:
                              cv2.rectangle(frame, (startX,
startY), (endX, endY),
                                       COLORS[idx], 2)
                              y = startY - 15 if startY - 15 >
15 else startY + 15
                              cv2.putText(frame, label, (startX,
y),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, COLORS[idx],
2)
                      if confidence<0.60:

                              break
                      if CLASSES[idx]=="bottle":
                              print("encontro botella") ##Bottle
                              botella()
```

```
                        elif CLASSES[idx]=="person":# import the
necessary packages
from imutils.video import VideoStream
from imutils.video import FPS
from multiprocessing import Process
from multiprocessing import Queue
import numpy as np
import argparse
import imutils
import time
import cv2
import getch
import time
import RPi.GPIO as GPIO


########################## Pines PWM
#########################################
GPIO.setmode(GPIO.BOARD)
servoPin1=35 #Pulgar.
servoPin2=33 #Doblar pulgar e indice.
servoPin3=12 #Dedo Medio.
servoPin4=36 #Anular y Menhique

GPIO.setup(servoPin1, GPIO.OUT)

GPIO.setup(servoPin2,GPIO.OUT)
GPIO.setup(servoPin3,GPIO.OUT)
GPIO.setup(servoPin4,GPIO.OUT)

GPIO.setwarnings(False)

pwm1 =GPIO.PWM(servoPin1,50)
pwm2 =GPIO.PWM(servoPin2,50)
pwm3 =GPIO.PWM(servoPin3,50)
pwm4 =GPIO.PWM(servoPin4,50)
############################################################
pwm1.start(0)
pwm2.start(0)
pwm3.start(0)
pwm4.start(0)
############# Functions Definition #################
def ceros():
    pwm1.ChangeDutyCycle(2.83)
    pwm2.ChangeDutyCycle(2.83)
    pwm3.ChangeDutyCycle(2.83)
    pwm4.ChangeDutyCycle(2.83)
def botella():
    DC1=1./18.*(60)+2
    pwm1.ChangeDutyCycle(DC1)
```

```
    ##
    DC2=1./18.*(150)+2
    pwm2.ChangeDutyCycle(DC2)
    ##
    DC3=1./18.*(150)+2
    pwm3.ChangeDutyCycle(DC3)
    ##
    DC4=1./18.*(160)+2
    pwm4.ChangeDutyCycle(DC4)
def persona():
    ceros()
    DC1=1./18.*(0)+2
    pwm1.ChangeDutyCycle(DC1)

    DC2=1./18.*(160)+2
    pwm2.ChangeDutyCycle(DC2)

    DC3=1./18.*(160)+2
    pwm3.ChangeDutyCycle(DC3)

    DC4=1./18.*(160)+2
    pwm4.ChangeDutyCycle(DC4)
    time.sleep(1.0)
    pwm1.ChangeDutyCycle(2.83)
    pwm2.ChangeDutyCycle(2.83)
    pwm3.ChangeDutyCycle(2.83)
    pwm4.ChangeDutyCycle(2.83)
    time.sleep(1.0)

    DC1=1./18.*(0)+2
    pwm1.ChangeDutyCycle(DC1)

    DC2=1./18.*(160)+2
    pwm2.ChangeDutyCycle(DC2)

    DC3=1./18.*(160)+2
    pwm3.ChangeDutyCycle(DC3)

    DC4=1./18.*(160)+2
    pwm4.ChangeDutyCycle(DC4)
    time.sleep(1.0)
    pwm1.ChangeDutyCycle(2.83)
    pwm2.ChangeDutyCycle(2.83)
    pwm3.ChangeDutyCycle(2.83)
    pwm4.ChangeDutyCycle(2.83)


################### Take a picture ###################
print('Press "a" to capture')
char=getch.getch()
```

```
if char=="a":
   camera=cv2.VideoCapture(0)
   return_value, vs=camera.read()
   cv2.imwrite('bot'+str(1)+'.jpg', vs)
   del(camera)


ceros()
def classify_frame(net, inputQueue, outputQueue):
      # keep looping
      while True:
            # check to see if there is a frame in our input queue
            if not inputQueue.empty():
                  # grab the frame from the input queue, resize it,
and
                  # construct a blob from it

                  frame = inputQueue.get()
                  frame=cv2.imread("bot1.jpg")
                  frame = cv2.resize(frame, (300, 300))
                  blob = cv2.dnn.blobFromImage(frame, 0.007843,
                        (300, 300), 127.5)

                  # set the blob as input to our deep learning
object
                  # detector and obtain the detections
                  net.setInput(blob)
                  detections = net.forward()

                  # write the detections to the output queue
                  outputQueue.put(detections)

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-p", "--prototxt", required=True,
      help="path to Caffe 'deploy' prototxt file")
ap.add_argument("-m", "--model", required=True,
      help="path to Caffe pre-trained model")
ap.add_argument("-c", "--confidence", type=float, default=0.2,
      help="minimum probability to filter weak detections")
args = vars(ap.parse_args())

# initialize the list of class labels MobileNet SSD was trained to
# detect, then generate a set of bounding box colors for each
class
CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",
      "bottle", "bus", "car", "cat", "chair", "cow", "diningtable",
```

```
	"dog", "horse", "motorbike", "person", "pottedplant",
"sheep",
	"sofa", "train", "tvmonitor"]
COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))

# load our serialized model from disk
print("[INFO] loading model...")
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])

# initialize the input queue (frames), output queue (detections),
# and the list of actual detections returned by the child process
inputQueue = Queue(maxsize=1)
outputQueue = Queue(maxsize=1)
detections = None

# construct a child process *indepedent* from our main process of
# execution
print("[INFO] starting process...")
p = Process(target=classify_frame, args=(net, inputQueue,
	outputQueue,))
p.daemon = True
p.start()

# initialize the video stream, allow the cammera sensor to warmup,
# and initialize the FPS counter
print("[INFO] starting video stream...")
frame=cv2.imread("bot1.jpg")
# vs = VideoStream(usePiCamera=True).start()
time.sleep(2.0)
fps = FPS().start()

# loop over the frames from the video stream
while True:
	# grab the frame from the threaded video stream, resize it,
and
	# grab its imensions
	frame=cv2.imread("bot1.jpg")
	frame = imutils.resize(frame, width=400)
	(fH, fW) = frame.shape[:2]

	# if the input queue *is* empty, give the current frame to
	# classify
	if inputQueue.empty():
		inputQueue.put(frame)
	# if the output queue *is not* empty, grab the detections
	if not outputQueue.empty():
		detections = outputQueue.get()
			print("entro")
	# check to see if our detectios are not None (and if so,
we'll
```

```
     # draw the detections on the frame)
     if detections is not None:
          # loop over the detections
          for i in np.arange(0, detections.shape[2]):
               # extract the confidence (i.e., probability)
associated
               # with the prediction
               confidence = detections[0, 0, i, 2]

               # filter out weak detections by ensuring the
`confidence`
               # is greater than the minimum confidence
               if confidence < args["confidence"]:
                    continue

               # otherwise, extract the index of the class label
from
               # the `detections`, then compute the (x, y)-
coordinates
               # of the bounding box for the object
               idx = int(detections[0, 0, i, 1])
               dims = np.array([fW, fH, fW, fH])
               box = detections[0, 0, i, 3:7] * dims
               (startX, startY, endX, endY) = box.astype("int")

               # draw the prediction on the frame

               label = "{}: {:.2f}%".format(CLASSES[idx],
                    confidence * 100)
               if confidence>0.60:
                                cv2.rectangle(frame, (startX,
startY), (endX, endY),
                                        COLORS[idx], 2)
                                y = startY - 15 if startY - 15 >
15 else startY + 15
                                cv2.putText(frame, label, (startX,
y),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, COLORS[idx],
2)
                      if confidence<0.60:

                                break
                      if CLASSES[idx]=="bottle":
                                print("encontro botella") ##Bottle
                                botella()
                      elif CLASSES[idx]=="person":
                                print("encontro persona") ##Person
                                persona()

     # show the output frame
```

```
      cv2.imshow("Frame", frame)
      key = cv2.waitKey(1) & 0xFF

      # if the `q` key was pressed, break from the loop
      if key == ord("q"):
            cv2.waitKey(0)
            cv2.destroyAllWindows()
            vs.stop()
            ceros()
         break

      # update the FPS counter
      fps.update()

# stop the timer and display FPS information
fps.stop()
print("[INFO] elapsed time: {:.2f}".format(fps.elapsed()))
print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))
# do a bit of cleanup
pwm1.stop()
pwm2.stop()
pwm3.stop()
pwm4.stop()
GPIO.cleanup()
```

```
                              print("encontro persona") ##Person
                              persona()

      # show the output frame
      cv2.imshow("Frame", frame)
      key = cv2.waitKey(1) & 0xFF

      # if the `q` key was pressed, break from the loop
      if key == ord("q"):
            cv2.waitKey(0)
            cv2.destroyAllWindows()
            vs.stop()
            ceros()
         break

      # update the FPS counter
      fps.update()

# stop the timer and display FPS information
fps.stop()
```

```
print("[INFO] elapsed time: {:.2f}".format(fps.elapsed()))
print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))
# do a bit of cleanup
pwm1.stop()
pwm2.stop()
pwm3.stop()
pwm4.stop()
GPIO.cleanup()
```

**Apéndice B.**

# Activities report.

## Camilo.

**Main objective:**

- **Implement an artificial intelligence algorithm in an embedded system to control a robotic arm capable of recognizing and taking objects.**

**Specific objectives:**

- **Determine the best control algorithm according to the characteristics of the embedded system.**
- **Perform the control of the actuators and sensors that are involved in the kinematics of the robotic hand.**

**Progress:** Install versions of the artificial vision libraries for the correct functioning on the embedded system.

Realization of algorithms that allow the handling of inputs and outputs of the embedded system as well as the calibration of the camera for taking image and determinate distance between the object and camera.

Support with CAD development and robotic hand printing with 3D printers.

## Harlinsson Chavarro:

**Main Objective**

**Evaluate different architectures about deep learning that allow recognize the detection of an object for its classification where this were compatible with raspberry pi device.**

**Specific objetives:**

- **Compare between different architecture about deep learning in raspberry pi.**
- **Implementation the architectures on raspberry pi device**

**Progress:**

Research about deep learning and application

Research about Theano, Keras, Tensorflow and CNTK

Install versions of the artificial vision libraries for the correct functioning on the embedded system.

Test of a model about library caffe where it can recognize many things, since an animal until kitchen things. In this case el recognize is fast, because it uses only one image for applying this architecture. We should know raspberry pi device has some limit in its memory and processing, therefore this model is a good option for apply.

FIRMA ESTUDIANTES Juan Camilo Cuervo Rpo.

HARLINSSON JAVIER CHAVARRO H.

FIRMA ASESOR _____

FIRMA ASESOR _____

FECHA ENTREGA: _____

FIRMA COMITÉ TRABAJO DE GRADO DE LA FACULTAD _____

RECHAZADO___ ACEPTADO____ ACEPTADO CON MODIFICACIONES_____

ACTA NO._____

FECHA ENTREGA: _____

FIRMA CONSEJO DE FACULTAD_____

ACTA NO._____

FECHA ENTREGA: _____