UNIVERSIDAD NACIONAL DEL CALLAO
ESCUELA DE POSGRADO

UNIDAD DE POSGRADO DE LA FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

“AUTOMATIZACIÓN DEL PROCESO DE SELECCIÓN DE MANGOS UTILIZANDO PROCESAMIENTO DE IMÁGENES Y REDES NEURONALES”

TESIS PARA OPTAR EL GRADO ACADÉMICO DE MAESTRO EN CIENCIAS DE LA ELECTRÓNICA CON MENCIÓN EN CONTROL Y AUTOMATIZACIÓN

AUTORES: FRANZUA LE RENNARD, OBLITAS ARISTONDO JAVIER AUGUSTO, CARRASCO JIMENEZ

ASESOR: DR. NICANOR RAÚL BENITES SARAVIA

LINEA DE INVESIGACIÓN: INGENIERÍA Y TECNOLOGIA

Callao, 2024

PERÚ

Curiginal

Document Information

Analyzed document	tesis_maestria.docx (D117444782)
Submitted	2021-11-04 15:56:00
Submitted by	
Submitter email	franzua.le@gmail.com
Similarity	13%
Analysis address	fiee.posgrado.unac@analysis.urkund.com

Sources included in the report

SA	**Universidad Nacional del Callao / AUTOMATIZACIÓN DEL PROCESO DE SELECCIÓN DE MANGOS UTILIZANDO PROCESAMIENTO DE IMÁGENES Y REDES NEURONALES.docx** Document AUTOMATIZACIÓN DEL PROCESO DE SELECCIÓN DE MANGOS UTILIZANDO PROCESAMIENTO DE IMÁGENES Y REDES NEURONALES.docx (D54478904) Submitted by: franzua.le@gmail.com Receiver: posgrado.fiee.unac@analysis.urkund.com	28
W	URL: http://lindaodettebenitezrosas1.blogspot.com/p/color.html Fetched: 2021-11-04 16:07:00	1
W	URL: https://es.wikipedia.org/wiki/Modelo_de_color_HSV Fetched: 2021-11-04 16:07:00	2
SA	**Universidad Nacional del Callao / AUTOMATIZACIÓN DEL PROCESO DE SELECCIÓN DE MANGOS UTILIZANDO PROCESAMIENTO DE IMÁGENES Y REDES NEURONALES.docx** Document AUTOMATIZACIÓN DEL PROCESO DE SELECCIÓN DE MANGOS UTILIZANDO PROCESAMIENTO DE IMÁGENES Y REDES NEURONALES.docx (D54296319) Submitted by: franzua.le@gmail.com Receiver: posgrado.fiee.unac@analysis.urkund.com	1

INFORMACIÓN BASICA

FACULTAD: Ingeniería Eléctrica y electrónica

UNIDAD DE INVESTIGACIÓN: POSGRADO – Facultad de ingeniería eléctrica y electrónica

Título: “Automatización del proceso de selección de mangos utilizando procesamiento de imágenes y redes neuronales”

AUTORES/DNI:

Ing. Franzua Le rennard Oblitas Aristondo/ DNI: 47902041

Bachiller Javier Augusto, Carrasco Jimenez/ DNI: 45525211

ASESOR/CODIGO ORCID/DNI:

Dr. Nicanor Raúl Benites Saravia/ CODIGO ORCID: 0000-0002-3245-6470/ DNI: 10189914

LUGAR DE EJECUCIÓN: Hacienda TamboGrande

TIPO/ ENFOQUE/ DISEÑO DE INVESTIGACIÓN:

Investigación de tipo aplicada, cuantitativo y diseño experimental

TEMA OCDE:

2.2.2 Maestría en Ciencias de la Electrónica

HOJA DE REFERENCIA DEL JURADO Y APROBACIÓN

DR. JACOB ASTOCONDOR VILLAR :PRESIDENTE

DR. FERNANDO MENDOZA APAZA :SECRETARIO

DR. ABILIO BERNARDINO CUZCANO RIVAS :MIEMBRO

MSC. CARLOS HUMBERTO ALFARO RODRIGUEZ :MIEMBRO

DR. NICANOR RAÚL BENITES SARAVIA :ASESOR

ACTA: 02 – UPG- 2024

LIBRO: 01

FOLIO 138

FECHA DE SUSTENTACIÓN 29 DE ENERO 2024

DEDICATORIA

A DIOS: Por darnos la fortaleza y la sabiduría para terminar con éxito nuestra tesis.

A mis padres, Norbil Alain Oblitas Campos y Milagros Rogelia Aristondo Chavez, mi hermano Jhon Andre Oblitas Aristondo, por el apoyo brindado y hacerme notar que siempre cuento con ellos.

A mi esposa y compañera de vida Rosa Zuleyka Elias Vera, por ser mi apoyo moral, motivación e inspiración

A mi amigo Javier Carrasco Jimenez, por su apoyo y conocimiento durante este trabajo.

A mi amigo Juan Abdul Salcedo Roman, por su apoyo moral a lo largo de nuestro trabajo

Oblitas Aristondo, Franzua Le rennard

Primeramente, dedico este trabajo de investigación al creador de todas las cosas, el que me ha dado fortaleza en las adversidades; y me ha permitido terminar con éxito esta tesis

De igual forma, dedico este trabajo a mis padres: Javier Adberto Carrasco Zapata y Dina Elsa Jimenez Rondan que me han inculcado conservar los valores y virtudes, los cuales me ha ayudado a salir adelante en los momentos más difíciles.

A mi compañera de vida Janeth Silva Valerio y mi pequeño hijo Eliel Carrasco Silva, por ser el motivo más grande que me impulsa a crecer día a día.

Carrasco Jimenez, Javier Augusto

ÍNDICE

TABLA DE CONTENIDOS

CONTENIDO DE FIGURAS

CONTENIDOS DE TABLAS

RESUMEN

Nos ubicamos en el distrito de TAMBOGRANDE, provincia de Piura departamento de Piura donde la familia Carrasco cuenta con 6 hectáreas de cultivo de mango, los cuales después de ser recolectados deben de ser clasificados en 2 grupos: los de consumo local y los que son para exportar, esta clasificación se realiza basado en 2 parámetros el color y su peso, los mangos de consumo local se caracterizan por tener un color rojizo o amarillento y un peso menor a 250gr y los mangos que son para exportar se caracterizan por tener un color verdoso y tener un peso mayor de 250gr.

La presente tesis titulada "AUTOMATIZACION DEL PROCESO DE SELECCIÓN DE MANGOS UTILIZANDO PROCESAMIENTO DE IMÁGENES Y REDES NEURONALES", nació de la necesidad de contar con un sistema de fácil configuración para hacer frente a la problemática del error humano que existe al momento de clasificar los mangos, lo cual ocasiona descontento entre los clientes

El sistema está conformado por una faja transportadora que cuenta con una balanza incorporada para reconocer el peso, un webcam para el reconocimiento de color y un servo motor para poder clasificar los mangos, adicionalmente el sistema cuenta con una interfaz gráfica desarrollada en el software libre PYTHON que nos permite reconocer el color aplicando técnica de procesamiento de imágenes (transformación de espacio vectorial RBG a HSV) y una neurona perceptrón que tiene como entrada 2 variables: peso y color, y una salida 0 (consumo local) y 1 (consumo extranjero).

Con la implementación de nuestra tesis se logra optimizar los tiempos en el proceso de selección de mangos, se obtiene una clasificación más justa y se evita el descontento entre los clientes.

RESUMO

Nos ubicamos no distrito de TAMBOGRANDE, província de Piura, departamento de Piura, onde a família Carrasco conta com 6 hectares de cultivo de manga, os quais depois de serem coletados devem ser classificados em 2 grupos: los de consumo local e los que son para exportar, esta classificação é realizada com base em 2 parâmetros na cor e no peso, as mangas de consumo local são caracterizadas por terem uma cor vermelha ou amarela e um peso menor a 250gr e as mangas que são para exportar são caracterizadas por ter uma cor verde e tem um peso maior de 250gr.

A presente tese intitulada “AUTOMATIZAÇÃO DO PROCESSO DE SELEÇÃO DE MANGOS UTILIZANDO PROCESSAMENTO DE IMAGENS E REDES NEURONAIS”, nasceu da necessidade de contar com um sistema de fácil configuração para fazer frente ao problema do erro humano que existe no momento de classificar as mangas , o que causa descontentamento entre os clients.

O sistema é composto por uma esteira transportadora que possui balança embutida para reconhecer o peso, uma webcam para reconhecimento de cores e um servo motor para classificação das mangas.Além disso, o sistema possui uma interface gráfica desenvolvida no software livre PYTHON que permite reconhecer a cor aplicando técnica de processamento de imagem (transformação do espaço vetorial RBG para HSV) e um neurônio perceptron que tem como entrada 2 variáveis: peso e cor, e uma saída 0 (consumo local) e 1 (consumo estrangeiro).

Com a implementação da nossa tese é possível otimizar tempos no processo de seleção das mangas, obter uma classificação mais justa e evitar a insatisfação dos clientes.

INTRODUCCIÓN

La familia Carrasco cuenta con una huerta en la hacienda de Tambo Grande, donde se dedican a cosechar gran cantidad de mangos, que luego son distribuidos en 2 tipos de mercado: Local y Extranjero. El problema que ellos presentan es al momento de clasificar estos mangos y para esto ellos han contratado trabajadores, pero aun así siguen teniendo problemas.

Esta investigación, es la automatización del proceso de selección de mangos utilizando técnicas de procesamiento de imágenes como transformación de espacio vectorial RGB a HSV y detección de bordes para determinar el color predominante en el mango y redes neuronales en donde creamos una neurona perceptrón con momentum cuyos valores de entradas serán el color y el peso y cuya salida es el destino final del mango (1 extranjero y 0 local).

Debido al gran tamaño del problema realizamos un prototipo que consiste en una caja de acero que en la parte superior tiene una webcam y una tira de leds para mantener constante el ambiente y poder realizar una buena etapa de procesamiento de imágenes y en la parte inferior cuenta con una balanza conectada a un sistema electrónico para visualizar el peso del mango y para transmitir dicho dato al Python, de esta manera se logra alimentar a nuestra neurona y así se logra automatizar el proceso de selección de mangos.

I PLANTEAMIENTO DEL PROBLEMA

1.1 Descripción de la realidad problemática

El error humano que existe al momento de clasificar los mangos

El problema que exponemos se presenta en todas las empresas exportadoras de mangos, tienen gran volumen de mangos, pero tienen problemas al momento de clasificarlos. Las empresas utilizan 2 parámetros para clasificar los mangos: Color y Peso (véase en la Figura N° 1.1). Para el proceso de clasificación utilizan un trabajador el cual detecta el color mediante la vista y el peso mediante una balanza. Los mangos de color verde-amarillento y que cuenten con un peso regular son clasificados como mangos para exportar debido a que demoran un tiempo para llegar los mercados extranjeros, tiempo que le sirve al mango para madurar y los mangos de color rojo-amarillento y que cuenten con un peso regular son clasificado como mangos para consumo local. Muchas veces por el gran volumen de mangos los trabajadores que se encargan de clasificar los mangos se confunden los cual genera queja y descontento entre sus clientes porque en el caso de los mercados extranjeros llegan algunos mangos podridos y en el caso de los mercados locales llegan algunos mangos que aún le faltan varios días para madurar.

Figura 1. Mango para mercado local y mango para mercado extranjero
Fuente: UNAC (2018); autoría propia

1.2 Formulación del problema

1.2.1 Problema General

¿De qué manera podemos automatizar el proceso de selección de mangos utilizando procesamiento de imágenes y redes neuronales?

1.2.2 Problema Específicos

Problema Especifico N°1: ¿Cómo podemos importar los valores de una balanza electrónica a Python?

Problema Especifico N°2: ¿De qué manera podemos mantener constante el ambiente para la captura de imágenes?

Problema Especifico N°3: ¿De qué manera podemos reconocer el color del mango e importarlos a Python?

Problema Especifico N°4: ¿De qué manera podemos utilizar los parámetros obtenidos (Color y Peso) para clasificar los mangos?

1.3 Objetivos de la investigación

1.3.1 Objetivo General

Desarrollar un algoritmo de control que permita automatizar la clasificación de los mangos en 2 tipos (Mercado Local y Mercado Extranjero) mediante la aplicación de una Neurona Perceptrón y utilizando como señales de entrada: el color y peso.

1.3.2 Objetivos Específicos

Objetivo Especifico N°1: Implementar un circuito electrónico mediante un Arduino UNO y un módulo HX710B para poder extraer de manera paralela los valores de una balanza, para luego importarlos a Python.

Objetivo Especifico N°2: Implementar un prototipo de hardware a escala para la captura de imágenes y así poder mantener el ambiente constante.

Objetivo Especifico N°3: Desarrollar un código de programación en Python para reconocer los colores del mango utilizando procesamiento de imágenes.

Objetivo Especifico N°4: Crear una Neurona Perceptrón en Python para clasificar los mangos en 2 tipos para mercados extranjeros y mercados locales y cuyos valores de entrada serán el color y el peso.

1.4 Limitantes de la investigación

1.4.1 Limitación teórica

Como limitación teórica se establece que debido a que no se cuenta con fuentes de información suficiente sobre la intensidad de color y peso de los mango, es necesario visitar las instalaciones del distrito de Tambo grande y seleccionar un grupo de 15 mangos de consumo local y 15 mangos para exportar, someterlos a pruebas con una balanza para obtener datos de peso y con el algoritmo de reconocimiento de colores con la finalidad de obtener valiosa información que servirá como entrenamiento para nuestra neurona.

1.4.2 Limitación temporal

Para la presente investigación nuestra limitación temporal se presentó principalmente al momento de importar las piezas para realizar un circuito electrónico que pueda extraer los valores de la balanza ya que dichos no son comerciales aquí en el mercado local, los elementos demorar aproximadamente entre 2 y 3 meses para llegar al Perú.

1.4.3 Limitación Espacial

Como limitación espacial podríamos decir que la presente investigación será diseñada para mejorar el proceso de selección de mangos que anteriormente realizaban los trabajadores específicamente del distrito de Tambo grande y desconocemos los resultados que podría tener con los mangos de otros distritos, porque para el aprendizaje nos basamos en

parámetros de peso y color de los mangos de Tambo grande, claro está que como es un trabajo de carácter experimental se pueden realizar nuevas pruebas para obtener nueva información para el entrenamiento de la neurona.

II MARCO TEÓRICO

2.1 Antecedentes del estudio

2.1.1 Antecedentes Nacionales

Como antecedentes o datos vinculados podemos mencionar:

Caso 1_ Diseño e Implementación de un Sistema de Control de Balanzas Industriales para Optimizar el Rebose de los Contenedores de la Empresa Mondelez.

Trabajo de tesis para optar el grado de Ingeniero Electrónico, sustentado por Oblitas Aristondo Franzua Le rennard y Salcedo Román Juan Abdul de la Universidad Nacional del Callao (UNAC, 2018). El presente trabajo de tesis tiene como objetivo crear un sistema de control que permita extraer de manera paralela datos de una balanza para luego ser comparado con un valor ingresado (Setpoint) y si el valor de la balanza es menor que el de referencia pues una electroválvula se abre y deja pasar la mezcla en los contenedores de caso contrario se cierra la electroválvula.

El problema que buscaba solucionar la empresa era el error Humano que existe al momento de llenar con mezcla los envases, ya que la empresa cuenta con una tolva gigante en el cual se juntan diversos insumos que son utilizados para generar una mezcla y luego esta mezcla servirá para producir diversos tipos de galleta, los insumos para la mezcla solo pueden ser subidos por un pequeño elevador y los trabajadores tienen que subir por una escalera para luego depositar los insumos en la tolva, una vez mezclado estos insumos el trabajador tiene que colocar un contenedor debajo de la tolva en el cual hay una llave de tipo manual y una balanza y el trabajador tiene que ir abriendo la llave hasta que el contenedor sea llenado con la cantidad de mezcla que estipula la receta, si en caso hubiera un exceso de mezcla el trabajador tiene que retirar ese exceso de mezcla y depositarlo nuevamente en la tolva, lo cual genera un doble esfuerzo para el trabajador y hace que se retrase el proceso de producción de galletas.

El objetivo del estudio era implementar un prototipo de sistema de control que pueda tomar decisiones de manera autónoma (On/ Off) sobre una electroválvula valiéndose del valor registrado en la balanza y no sobrepasando el valor ingresado en la Aplicación, con la finalidad de optimizar el rebose en los contenedores de la empresa Mondelez y así poder evitar un sobreesfuerzo humano por parte de los trabajadores, lograr una optimización en los tiempos de producción de las galletas y mejorar la calidad de sus productos.

Durante el proyecto se implementó un sistema de control embebido en el cual cuenta con una toma de 220v, un switch (On/Off), una conexión USB, un colector de 5 pines para celda de carga y un colector de 2 pines para la electroválvula (véase la Figura N°2).

Figura 2. Implementación del sistema de control de balanzas industriales.
Fuente: Perú (2018); Oblitas y Salcedo

2.1.2 Antecedentes Internacionales

Caso 1_ Transformación de Espacio Vectorial de RGB a HSV

Es un pequeño tutorial Desarrollado por Alexander Mordvintsev y Abid K. (Abid, 2013) Tutorial realizado para realizar de una manera más sencilla el seguimiento de las imágenes por el color pasando las imágenes de un

espacio vectorial RGB a HSV. Cada tonalidad de color está representado en un espacio tridimensional RGB, es decir cada color está conformado por un vector (Red, Green, Blue) lo cual dificulta su ubicación en el espacio, por lo cual es necesario llevarlo a otro tipo de espacio, para nuestro caso lo llevaremos al espacio vectorial HSV (Hue, Saturacion, Value) en donde el color se representa en un solo valor (value) y los otros 2 parámetros H y S solo representan el tono y la intensidad respectivamente (véase la Figura N°3).

```
import cv2
import numpy as np
cap = cv2.VideoCapture(0)
while(1):
    # Take each frame
    _, frame = cap.read()
    # Convert BGR to HSV
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    # define range of blue color in HSV
    lower_blue = np.array([110,50,50])
    upper_blue = np.array([130,255,255])
    # Threshold the HSV image to get only blue colors
    mask = cv2.inRange(hsv, lower_blue, upper_blue)
    # Bitwise-AND mask and original image
    res = cv2.bitwise_and(frame,frame, mask= mask)
    cv2.imshow('frame',frame)
    cv2.imshow('mask',mask)
    cv2.imshow('res',res)
    k = cv2.waitKey(5) & 0xFF
    if k == 27:
        break
cv2.destroyAllWindows()
```

Figura 3. Algoritmo para el cambio de espacio vectorial RGB a HSV.
Fuente: EE. UU (2013); Alexander Mordvintsev y Abid K.

Primero encendemos la webcam, luego hacemos la transformación del espacio vectorial RGB a HSV, en este caso realizaremos un seguimiento al color azul, para lo cual definiremos 2 matrices: azul bajo y azul alto (0 - 255) los cuales serán cargados en una máscara, así si la tonalidad de nuestra imagen se encuentra dentro de los límites de nuestra matriz este será seleccionado mientras que el resto será ignorado (véase la Figura N° 4).

Figura 4. Cambio de espacio vectorial RGB a HSV paso a paso.
Fuente: EE. UU (2013); Alexander Mordvintsev y Abid K.

Como se puede observar a pesar de que realizamos la transformación vectorial de RGB a HSV sigue apareciendo ruido, pero actualmente es la técnica más utilizada para seleccionar objetos por el color la cual luego será combinada con filtros morfológicos para eliminar el ruido.

Caso 2_ Creación de 6 Trackbars para la transformación de espacio vectorial RGB a HSV

Tutorial desarrollado por la pagina stackoverflow (stackoverflow, 2017) para solucionar el problema detectado en el caso 2, cada vez que queríamos modificar los valores HSV tenemos que cerrar todas la ventanas de imágenes, luego modificar los valores HSV y luego volver a correr el programa lo cual termina siendo muy aburrido y repetitivo, para lo cual se desarrolló un algoritmo que permite dibujar en una ventana aparte con 6 trackbars (H_{min}, S_{min}, V_{min}, H_{max}, S_{max}, V_{max}) los cuales nos permite variar los parámetros HSV sin necesidad de tener que cerrar ninguna ventana y los cambios se pueden apreciar en tiempo real. (Véase la Figura N°5)

```
import cv2
import numpy as np
cap = cv2.VideoCapture(0)
def nothing(x):
    pass
#Creamos una ventana llamada 'image' en la que habra todos los sliders
cv2.namedWindow('image')
cv2.createTrackbar('Hue Minimo','image',0,255,nothing)
cv2.createTrackbar('Hue Maximo','image',0,255,nothing)
cv2.createTrackbar('Saturation Minimo','image',0,255,nothing)
cv2.createTrackbar('Saturation Maximo','image',0,255,nothing)
cv2.createTrackbar('Value Minimo','image',0,255,nothing)
cv2.createTrackbar('Value Maximo','image',0,255,nothing)

while(1):
    _,frame = cap.read() #Leer un frame
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV) #Convertirlo a espacio de color HSV

    #Los valores maximo y minimo de H,S y V se guardan en funcion de la posicion de los sliders
    hMin = cv2.getTrackbarPos('Hue Minimo','image')
    hMax = cv2.getTrackbarPos('Hue Maximo','image')
    sMin = cv2.getTrackbarPos('Saturation Minimo','image')
    sMax = cv2.getTrackbarPos('Saturation Maximo','image')
    vMin = cv2.getTrackbarPos('Value Minimo','image')
    #Se crea un array con las posiciones minimas y maximas
    lower=np.array([hMin,sMin,vMin])
    upper=np.array([hMax,sMax,vMax])

    #Deteccion de colores
    mask = cv2.inRange(hsv, lower, upper)

    #Mostrar los resultados y salir
    cv2.imshow('camara',frame)
    cv2.imshow('mask',mask)

    k = cv2.waitKey(5) & 0xFF
    if k == 27:
      break
cv2.destroyAllWindows()
```

Figura 5. Algoritmo de transformación vectorial RGB a HSV utilizando Trackbars. Fuente: EE. UU (2017); Stackoverflow.

Iniciamos el programa encendiendo la webcam, luego dibujamos una ventana con el nombre “image” en la cual dibujaremos 6 trackbars (Hue Minimo, Hue Maximo, Saturation Minimo, Saturation Maximo, Value Minimo y Value Maximo) cuyos valores solo pueden varia de (0-255), luego ingresamos a bucle de tipo While en donde guardaremos las imágenes capturadas por la cámara en la variable “frame”, luego realizaremos una transformación de espacio vectorial RGB a HSV, luego capturaremos los 6 valores de los trackbars en las variables (hMin, hMax, sMin, sMax, Vmin y Vmax) los cuales son almacenados en 2 matrices (lower y upper), luego cargamos estas 2 matrices en nuestra imagen de formato HSV, luego mostramos los resultados en 2 ventanas (cámara y mask) en donde mostramos el frame y la máscara final respectivamente y finalmente

configuramos el programa para que se detenga cuando presionamos la tecla “ESC”. (Véase la Figura N°6)

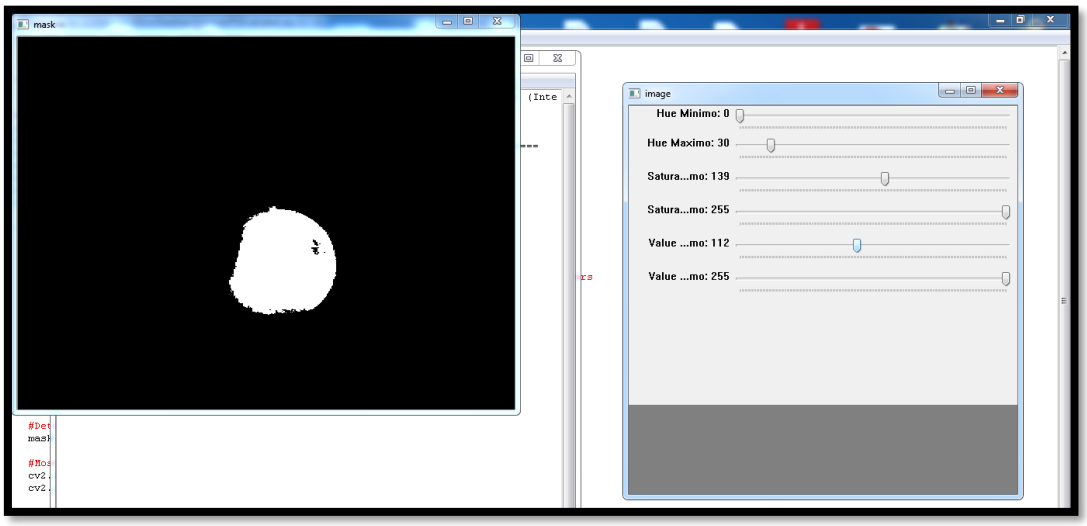


Figura 6. Detección de objetos mediante color.
Fuente: EE. UU (2017); Stackoverflow.

Caso 3_ Filtros Morfológicos Erosión y Dilatación

Tutorial desarrollado por Alexander Mordvintsev y Abid K. (Abid, 2013) con la finalidad de eliminar el ruido que se presenta en una imagen, para esto aplicamos el filtro de erosión que consiste en definir una matriz cuadra e impar llamada kernel conformada por números uno, el elemento central del kernel se posiciona en un inicio el primer pixel de la imagen y si todos los elementos que se encuentra alrededor son unos este pixel se mantiene como uno, de caso contrario se convertirá en cero, de esta manera se logra eliminar el ruido presente en una imagen, pero a la vez también deforma los borde de la imagen, luego aplicamos el filtro de dilatación, utilizamos el mismo kernel que definimos en la erosión, el elemento central se posiciona en el primer pixel de la imagen y basta con que solo un elemento de los que está a su alrededor sea 1 ese pixel se convierte en 1, de esta manera se recupera la forma de la imagen y el ruido no crece puesto que ya fue eliminado con el filtro de erosión (Véase la Figura N°7)

```
import cv2
import numpy as np

img = cv2.imread('pelota.jpg',0)
kernel = np.ones((5,5),np.uint8)
erosion = cv2.erode(img,kernel,iterations = 2)
dilation = cv2.dilate(erosion,kernel,iterations = 3)
cv2.imshow('elimancion del ruido',dilation)
```

Figura 7. Programa para eliminar el ruido de una imagen.
Fuente: EE. UU (2013); Alexander Mordvintsev y Abid K.

Primero cargamos una imagen llamada “pelota” que esta en formato .jpg mediante el comando imread, al final agregamos “,0” para poder pasar la imagen de RGB a escala de grises, luego definimos una matriz cuadrada impar llamado kernel de 5x5 conformado por elementos 1, luego aplicamos el filtro de erosión mediante el comando erode, la ventaja de hacerlo en Python es que podemos indicar el número de veces que queremos que se aplique este filtro y luego aplicamos el filtro de dilatación mediante el comando dilate y finalmente mostramos el resultado. (Véase la Figura N°8)

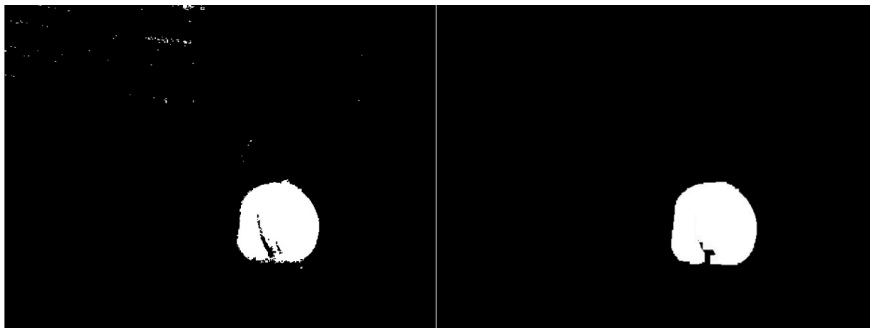
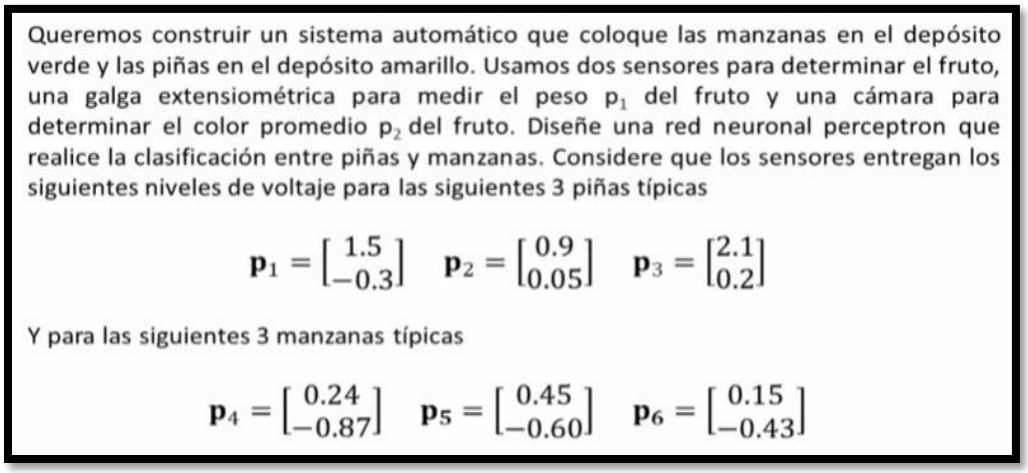


Figura 8. Imagen con y sin ruido.
Fuente: EE. UU (2013); Alexander Mordvintsev y Abid K.

Caso 4_ Clasificación de Piñas y Manzanas utilizando Redes Neuronales

Video Tutorial desarrollado por el canal de YouTube: HackeandoTec (HackeandoTec, 2016).El cual nos permite entender mejor como funciona una Neurona Perceptrón. El cual propone construir un sistema automático

que coloque las manzanas en el depósito verde y las piñas en el depósito amarillo. Utilizando como entradas parámetros como el Peso y el Color (véase la Figura N° 9).


Queremos construir un sistema automático que coloque las manzanas en el depósito verde y las piñas en el depósito amarillo. Usamos dos sensores para determinar el fruto, una galga extensiométrica para medir el peso p_1 del fruto y una cámara para determinar el color promedio p_2 del fruto. Diseñe una red neuronal perceptron que realice la clasificación entre piñas y manzanas. Considere que los sensores entregan los siguientes niveles de voltaje para las siguientes 3 piñas típicas

$$\mathbf{p}_1 = \begin{bmatrix} 1.5 \\ -0.3 \end{bmatrix} \quad \mathbf{p}_2 = \begin{bmatrix} 0.9 \\ 0.05 \end{bmatrix} \quad \mathbf{p}_3 = \begin{bmatrix} 2.1 \\ 0.2 \end{bmatrix}$$

Y para las siguientes 3 manzanas típicas

$$\mathbf{p}_4 = \begin{bmatrix} 0.24 \\ -0.87 \end{bmatrix} \quad \mathbf{p}_5 = \begin{bmatrix} 0.45 \\ -0.60 \end{bmatrix} \quad \mathbf{p}_6 = \begin{bmatrix} 0.15 \\ -0.43 \end{bmatrix}$$

Figura 9. Ejemplo para entender el funcionamiento de la neurona perceptrón
Fuente: Granada (2016); HackeandoTec

Modelo Estándar de una Neurona Artificial:

Considerando que la regla de propagación es la suma ponderada y que la función de salida es la identidad, la neurona estándar consiste en:

- Un conjunto de **ENTRADAS:** $X_j(t)$
- Unos **PESOS SINÁPTICOS** w_{ij} asociados a las entradas.
- Una **REGLA DE PROPAGACIÓN** $h_i(t) = \sigma(w_{ij}, x_j(t))$. La más común suele ser $h_i(t) = \sum w_{ij} x_j$
- Una **FUNCIÓN DE ACTIVACIÓN** $y_i(t) = f_i(h_i(t))$ que representa simultáneamente la salida de la neurona y su estado de activación.

Todos estos elementos quedan recogidos en la siguiente ilustración que pone de manifiesto el modelo considerado. (Véase Figura N° 10).

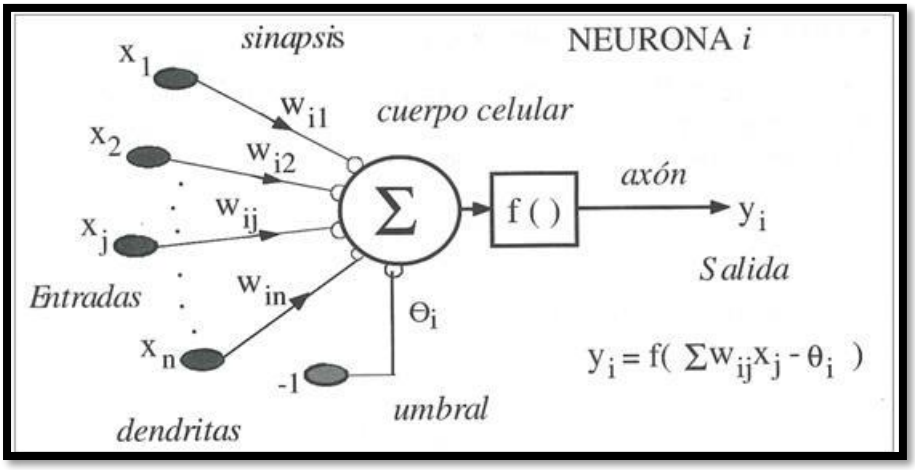


Figura 10. Modelo estándar de neurona artificial
Fuente: Granada (2016); HackeandoTec

Si $y_i > 1$ es clasificado como manzana de caso contrario es considerado Piña. Graficamos los pesos sinápticos (véase la Figura N° 11)

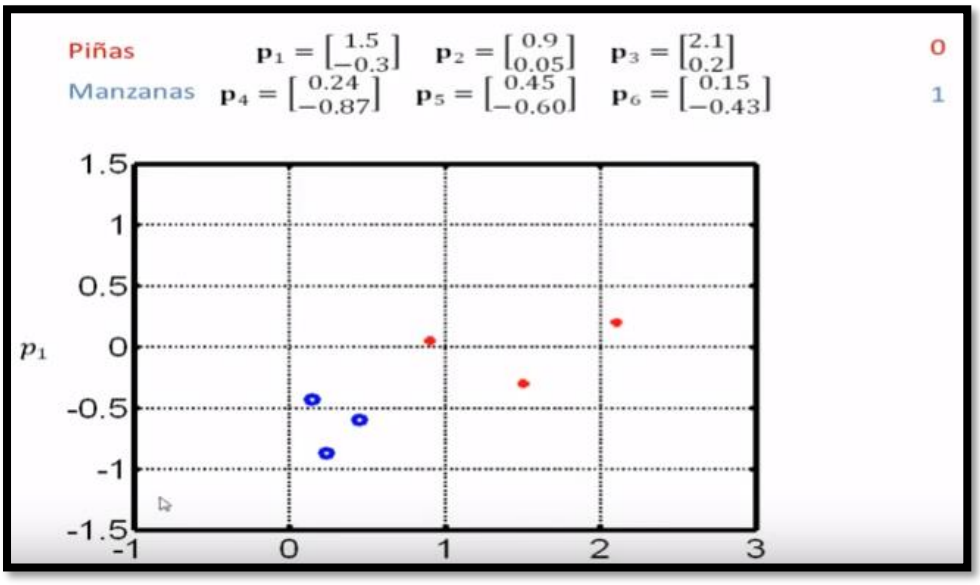


Figura 11. Grafica de los pesos sinápticos.
Fuente: Granada (2016); HackeandoTec

Trazamos una recta divisoria y calculamos la ecuación de la recta. (Véase Figura N° 12)

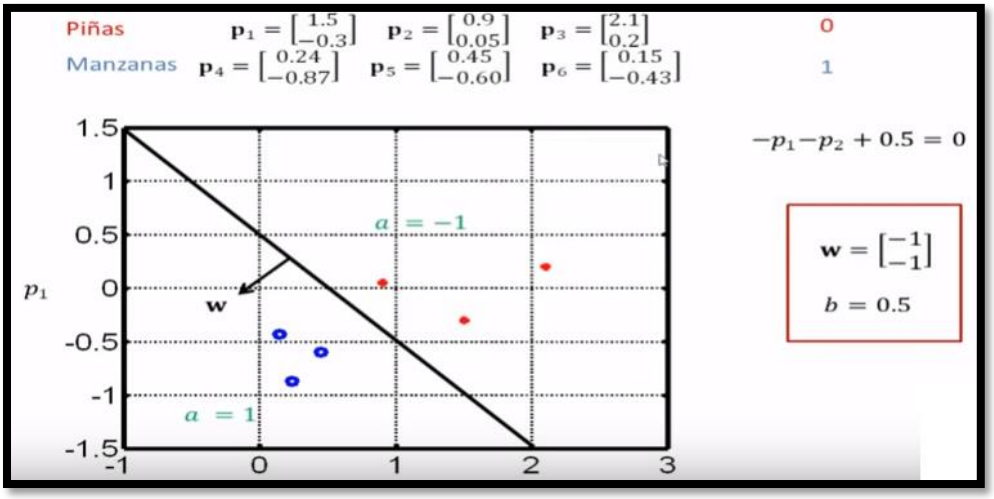


Figura 12. Grafica de la recta divisoria y cálculo de la ecuación de la recta
Fuente: Granada (2016); HackeandoTec

De la Figura N° 2.8 se obtiene lo siguiente como puede verse en la ecuación 1.

$$y = -p_1 - p_2 + 0.5 \quad \textbf{(1)}$$

Donde:

$y =$ Función de Activación

$p_1 =$ Entrada 1 (Peso de la Fruta)

$p_2 =$ Entrada 2 (Color de la Fruta)

Analicemos la función de activación en: $p = \begin{bmatrix} 1.5 \\ -0.3 \end{bmatrix}$ reemplazamos en la ecuación 1.

$$y = -1.5 - (-0.3) + 0.5$$

$$y = -0.8$$

Como la función de activación y es menor a 0 se deduce que la fruta es una piña como se explicó al comienzo del ejercicio.

Caso 5_ Determinación de los pesos sinápticos de una neurona perceptrón con momentum

La siguiente información fue obtenida de ("Red Neuronal de Topología Flexible", 1999). Entrenar una neurona con función de activación limitador duro para el aprendizaje de la función lógica OR que se muestra (véase la tabla N°2.1), donde las entradas y salidas son bipolares (aceleran el aprendizaje de la neurona). Utilizar el algoritmo de corrección de error α-LMS y tomar el vector de pesos inicial $[w^1]^T$ = [0.2 0.1 0.1], el factor de aprendizaje α = 0.1.

***TABLA 1**. Función lógica OR bipolar*

X_1	X_2	Y
-1	-1	-1
-1	+1	+1
+1	-1	+1
+1	+1	+1

Fuente: La Paz (1999); Aguilar, R.

Iteración (k=1)

Entradas $x_1^1 = -1, x_2^1 = -1$. Salida deseada $y_d^1 = -1$. La salida del combinador lineal s^1 es como puede verse en la ecuación 2.

$$s^1 = (0.2)(1) + (0.1)(-1) + (0.1)(-1) = 0 \quad \textbf{(2)}$$

$$y^1 = f(s^1) = 1$$

$$e^1 = y_d^1 - y^1 = (-1) - (+1) = -2$$

Actualizando el vector de pesos: como puede verse en la ecuación 3.

$$w^2 = w^1 + \alpha e^1 \frac{x^1}{\|x^1\|^2} \quad \textbf{(3)}$$

$$\begin{bmatrix} w_0^2 \\ w_1^2 \\ w_2^2 \end{bmatrix} = \begin{bmatrix} 0.2 \\ 0.1 \\ 0.1 \end{bmatrix} + 0.1(-2)\frac{1}{1^2 + (-1)^2 + (-1)^2}\begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.1333 \\ 0.1667 \\ 0.1667 \end{bmatrix}$$

Iteración (k=2)

Entradas $x_1^2 = -1, x_2^2 = +1$. Salida deseada $y_d^2 = +1$. La salida del combinador lineal s^2 es como puede verse en la ecuación 4.

$$s^2 = (0.1333)(1) + (0.1667)(-1) + (0.1667)(+1) = 0.1333 \quad \textbf{(4)}$$

$$y^2 = f(s^2) = +1$$

$$e^2 = y_d^2 - y^2 = (+1) - (+1) = 0$$

El nuevo vector de pesos es igual al anterior, como puede verse en la ecuación 5.

$$w^3 = w^2 \quad \textbf{(5)}$$

Iteración (k=3)

Entradas $x_1^3 = +1, x_2^3 = -1$. Salida deseada $y_d^3 = +1$. La salida del combinador lineal s^3 es como puede verse en la ecuación 6.

$$s^3 = (0.1333)(1) + (0.1667)(+1) + (0.1667)(-1) = 0.1333 \quad \textbf{(6)}$$

$$y^3 = f(s^3) = +1$$

$$e^3 = y_d^3 - y^3 = (+1) - (+1) = 0$$

El nuevo vector de pesos es igual al anterior. Como de observa en la ecuación 7.

$$w^4 = w^3 \quad \textbf{(7)}$$

Iteración (k=4)

Entradas $x_1^4 = +1, x_2^4 = +1$. Salida deseada $y_d^4 = +1$. La salida del combinador lineal s^4 es como puede verse en la ecuación 8.

$$s^4 = (0.1333)(1) + (0.1667)(+1) + (0.1667)(+1) = 0.4667 \quad \textbf{(8)}$$

$$y^4 = f(s^4) = +1$$

$$e^4 = y_d^4 - y^4 = (+1) - (+1) = 0$$

El nuevo vector de pesos es igual al anterior. Como de observa en la ecuación 9.

$$w^5 = w^4 = \begin{bmatrix} 0.1333 \\ 0.1667 \\ 0.1667 \end{bmatrix} \quad \textbf{(9)}$$

Iteración (k=5)

Entradas $x_1^5 = -1, x_2^5 = -1$. Salida deseada $y_d^5 = +1$. La salida del combinador lineal s^5 es como puede verse en la ecuación 10

$$s^5 = (0.1333)(1) + (0.1667)(-1) + (0.1667)(-1) = -0.2001 \quad \textbf{(10)}$$

$$y^5 = f(s^5) = -1$$

$$e^5 = y_d^5 - y^5 = (-1) - (-1) = 0$$

Como el error es cero significa que el vector de pesos asociado a la neurona satisface los patrones de la Tabla N° 2.1 de la función lógica OR. (Véase la Figura N° 13) muestra a la topología de la red con los pesos actualizados.

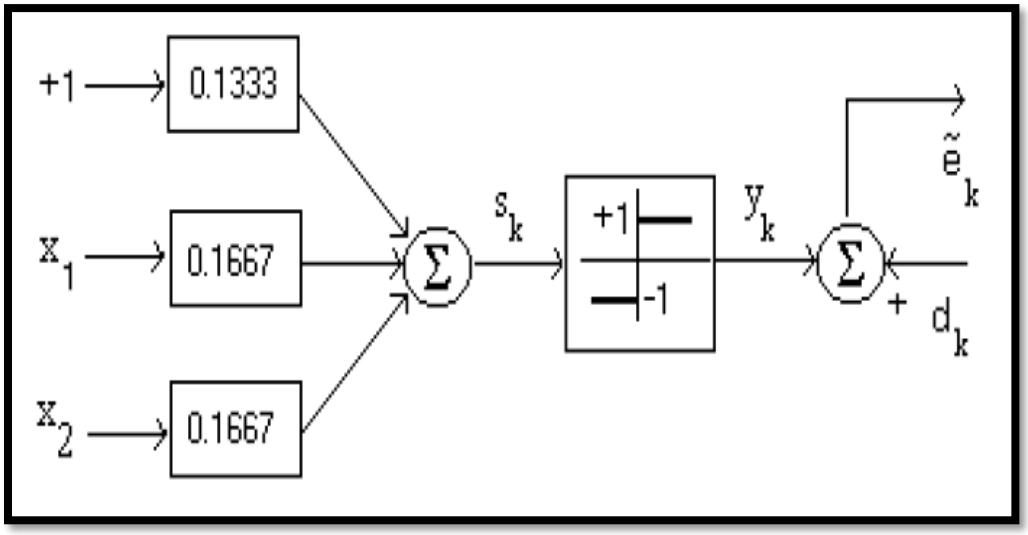


Figura 13. Red neuronal entrenada.
Fuente: La Paz (1999); Aguilar, R.

2.2 Base Teóricas

2.2.1 Base Epistémica

El desarrollo de algoritmos para el procesamiento de imágenes ha cobrado mayor impulso en el ámbito agrícola, que van desde la creación de un robot que puede eliminar la maleza en terrenos de cultivos aplicando algoritmos de procesamiento de imagen para diferenciar la planta original de la maleza hasta creación de algoritmos de imagen para reconocer frutas basado en parámetro como color y forma, lo que nuestra presente investigación propone es crear un algoritmo de reconocimiento de colores para diferenciar básicamente 2 colores el amarillo verdoso y amarillo rojizo los cuales y con dicha información someter a un entrenamiento a una neurona perceptrón y así esta pueda clasificar el mango entre dos tipos: Consumo Local y Exportación.

2.2.2 Base ontológica

Los sistemas de automatismo de control tienen el desafío de generar sus propios conocimientos a través de la investigación y construir desde su propia perspectiva. Los sistemas de redes neuronales se sustentan en conocimientos científicos como soporte al desarrollo del mismo.

Las tecnologías de procesamiento de imágenes y redes neuronales tienen implicaciones revolucionarias en terreno agrícola y en la vida cotidiana del ser humano, que gracias a la electrónica se incrementan a un de cambio nunca antes conocido.

Los sistemas de redes neuronales nos ayudan a evitar tareas que son repetitivas para el ser humano y por el mismo hecho de ser repetitiva se corre el riesgo de que ocurran errores. Las neuronas al igual que los seres humanos vienen vacías y al igual que nosotros requieren de un entrenamiento, para que luego puedan realizar una tarea basada en la experiencia del entrenamiento.

2.3 Marco conceptual

2.3.1 Celda de Carga

La siguiente información fue obtenida de la siguiente referencia (OMEGA, 2019). Una célula de carga (o celda de carga) es un transductor que convierte la fuerza aplicada sobre ella en una señal eléctrica medible. A pesar de existir varios tipos de sensores, las células de carga son el sensor de fuerza más común del mercado.

Los diseños de células de carga se pueden distinguir de acuerdo con el tipo de señal de salida generada (neumático, hidráulico, eléctrico) o de acuerdo con la forma en que detectan el peso (flexión, cizalladura, compresión, tensión, etc.). (Véase la Figura N° 14) podemos observar una galga de carga extraída de una balanza electrónica de laboratorio modelo SF-400

Figura 14. Celda de carga de 5kg
Fuente: España (2019); Omega

Puente de Wheatstone:

La siguiente información fue obtenida de la siguiente referencia (HBM, 2017). El puente de Wheatstone puede utilizarse de varias maneras para medir la resistencia eléctrica:

- para determinar el valor absoluto de una resistencia mediante comparación con otra resistencia conocida
- para determinar cambios relativos en la resistencia

Esta última aplicación es la que se utiliza con las galgas extensométricas, ya que permite medir con elevada exactitud cambios relativos en la resistencia de una galga extensométrica, normalmente del orden de entre 10-4 y 10-2 Ω/Ω. (Véase la Figura N° 15)

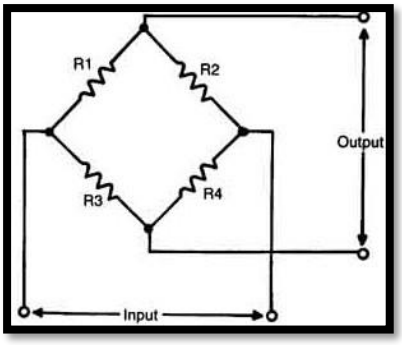


Figura 15. Puente de Wheatstone
Fuente: España (2017); HBM

Si conocemos los valores de las cuatro resistencias y la fuente de voltaje (E), y la resistencia del galvanómetro es lo suficientemente alta para que IE sea despreciable, el voltaje en el galvanómetro (VG) se puede determinar mediante la fórmula del puente de Wheatstone como puede verse en la ecuación 11

$$V_G = \left(\frac{R_3}{R_1+R_3} - \frac{R_4}{R_2+R_4}\right) \quad \textbf{(11)}$$

(Véase la Figura N° 16) podemos observar el código de colores en la galga extensiométrica, donde el cable negro y rojo son las Entradas y los cables verdes y blancos son las Salidas.

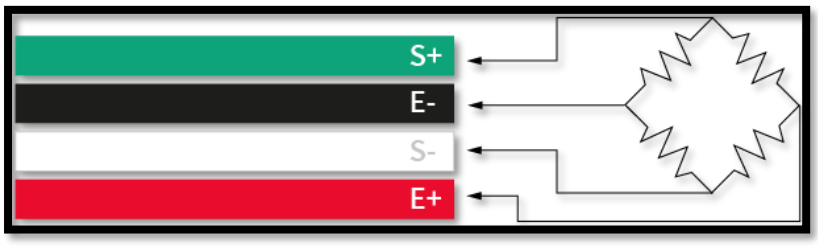


Figura 16. Código de colores de la celda de carga
Fuente: España (2017); HBM

2.3.2 Módulo HX710B

La siguiente información fue obtenida de la siguiente fuente: (Semiconductor, 2017) Basado en los productos patentados de Avía Semiconductor. La tecnología HX710 (A / B) es de una precisión de 24 bits. Convertidor analógico a digital (ADC) con un sensor incorporado de temperatura (HX710A) o DVDD, AVDD Detección de diferencia de tensión (HX710B). Diseñado para básculas y control industrial. Aplicaciones para interactuar directamente con un puente de Wheatstone. El amplificador de entrada de bajo ruido (PGA) tiene una ganancia fija de 128, correspondiente a una escala completa. Voltaje de entrada diferencial de ± 20mV, cuando 5V. La tensión de referencia está conectada al pin VREF. El chip del oscilador proporciona el reloj del sistema. Sin ningún componente externo. El circuito de reinicio de energía en el chip simplifica la interfaz digital inicialización No hay programación necesaria para los registros internos. Todos los controles al HX710 son a través de los pines. (Véase la Figura N° 17)

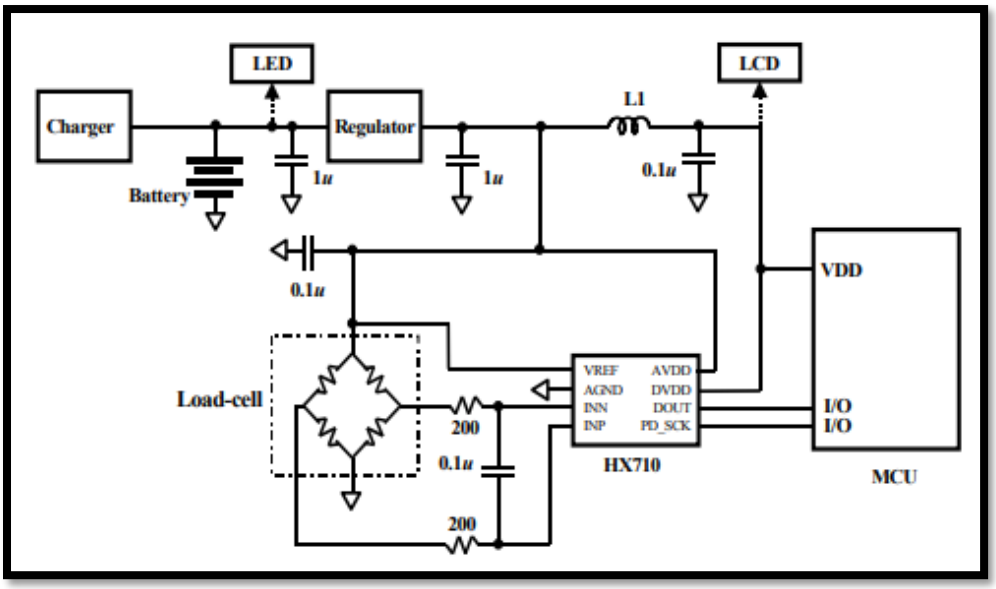


Figura 17. Diagrama de bloques típico de la aplicación de la báscula de pesaje con HX710B

Fuente: China (2017); Semiconductor

CARACTERISTICAS:

- Medición de temperatura en chip (HX701A)
- DVDD y AVDD diferencia de tensión de alimentación medida (HX701B)
- Amplificador de bajo ruido en chip con una ganancia de 128.
- Oscilador en chip que no requiere externo componente
- Power-on-reset en el chip
- Control digital simple e interfaz serial: Controles accionados por pin, sin necesidad de programación
- Velocidad de datos de salida seleccionable de 10SPS o 40SPS
- Rechazo de suministro simultáneo de 50 y 60Hz.
- Consumo actual: operación normal <1.2mA, apagado <1uA
- Rango de voltaje de alimentación de operación: 2.6 ~ 5.5V
- Rango de temperatura de operación: -40 ~ + 85
- Paquete de 8 pin SOP-8

El HX710 es un circuito integrado diseñado para balanzas o básculas electrónicas utilizando celda de carga (sensor bridge).

Estas celdas de carga son las más utilizadas en los sistemas de pesaje domésticos, comercios, así como los industriales. (Véase la Figura N° 18)

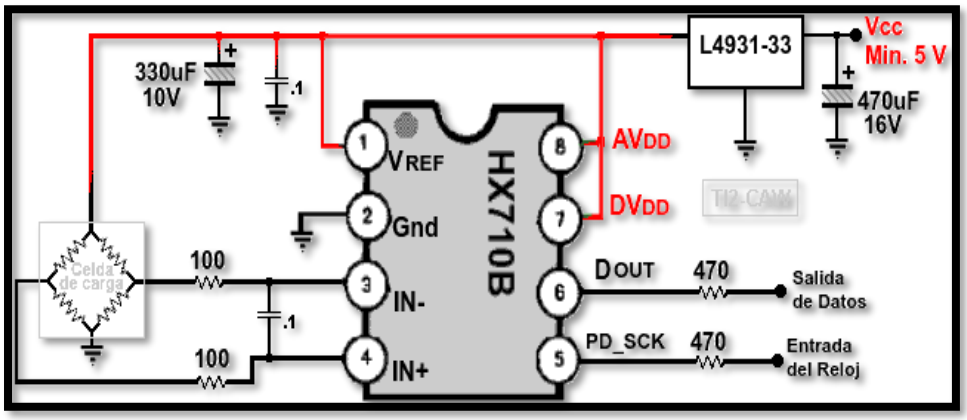


Figura 18. Conexiones entre modulo HX710B y la celda de carga.
Fuente: China (2017); Semiconductor

Las resistencias de 100 ohmios entre la celda de carga y el HX710 junto con un capacitor cerámico de 0,1uF (104) forman un filtro para limitar algunas interferencias, he visto de 47 ohmios en básculas comerciales y el fabricante muestra como ejemplo 200 ohmios.

El HX710A tanto como el HX710B pueden trabajar de 2.6V a 5.5V, la entrada de voltaje análoga AVDD no debe superar al voltaje digital DVDD

El voltaje de referencia VREF debe de ser superior a 1.8V y no superar el voltaje AVDD

Las patillas que van hacia el Arduino llevan una resistencia de 470 ohmios cada una, en los diagramas de ejemplo del fabricante no se muestran ya que no son indispensables, aunque yo las utilizo porque todas las básculas comerciales las tienen (HX710 a PIC), y como no estorban me parece que pueden proteger las entradas.

Yo regulo el voltaje aparte ya que no quiero dañar el regulador propio del Arduino en caso de un accidente con los cables. El mejor regulador es el L4931-33 ya que puede dar los 3.3 voltios a partir de 3.9 voltios, otros como el L78L33 necesitan muy cerca de 5 voltios para dar 3.3V de manera estable y soporta menos mA.

2.3.3 Python

La siguiente información fue obtenida de (Sphinx, 2017). Python es un lenguaje de programación poderoso y fácil de aprender. Cuenta con estructuras de datos eficientes y de alto nivel y un enfoque simple pero efectivo a la programación orientada a objetos. La elegante sintaxis de Python y su tipado dinámico, junto con su naturaleza interpretada, hacen de éste un lenguaje ideal para scripting y desarrollo rápido de aplicaciones en diversas áreas y sobre la mayoría de las plataformas.

El intérprete de Python y la extensa biblioteca estándar están a libre disposición en forma binaria y de código fuente para las principales plataformas desde el sitio web de Python, https://www.python.org/, y puede distribuirse libremente. El mismo sitio contiene también distribuciones y enlaces de muchos módulos libres de Python de terceros, programas y herramientas, y documentación adicional.

El intérprete de Python puede extenderse fácilmente con nuevas funcionalidades y tipos de datos implementados en C o C++ (u otros lenguajes accesibles desde C). Python también puede usarse como un lenguaje de extensiones para aplicaciones personalizables.

OpenCV

OpenCV es una biblioteca libre de visión artificial originalmente desarrollada por Intel. Desde que apareció su primera versión alfa en el mes de enero de 1999, se ha utilizado en infinidad de aplicaciones. Desde sistemas de seguridad con detección de movimiento, hasta aplicaciones de control de procesos donde se requiere reconocimiento de objetos. Esto se debe a que su publicación se da bajo licencia BSD, que permite que sea usada libremente para propósitos comerciales y de investigación con las condiciones en ella expresadas.

Open CV es multiplataforma, existiendo versiones para GNU/Linux, Mac OS X, Windows y Android. Contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos (reconocimiento facial), calibración de cámaras, visión estérea y visión robótica.

El proyecto pretende proporcionar un entorno de desarrollo fácil de utilizar y altamente eficiente. Esto se ha logrado realizando su programación en código C, C++ y Python optimizados, aprovechando además las capacidades que proveen los procesadores multinúcleo. OpenCV puede además utilizar el sistema de primitivas de rendimiento integradas de Intel,

un conjunto de rutinas de bajo nivel específicas para procesadores Intel (Véase la Figura N° 19)

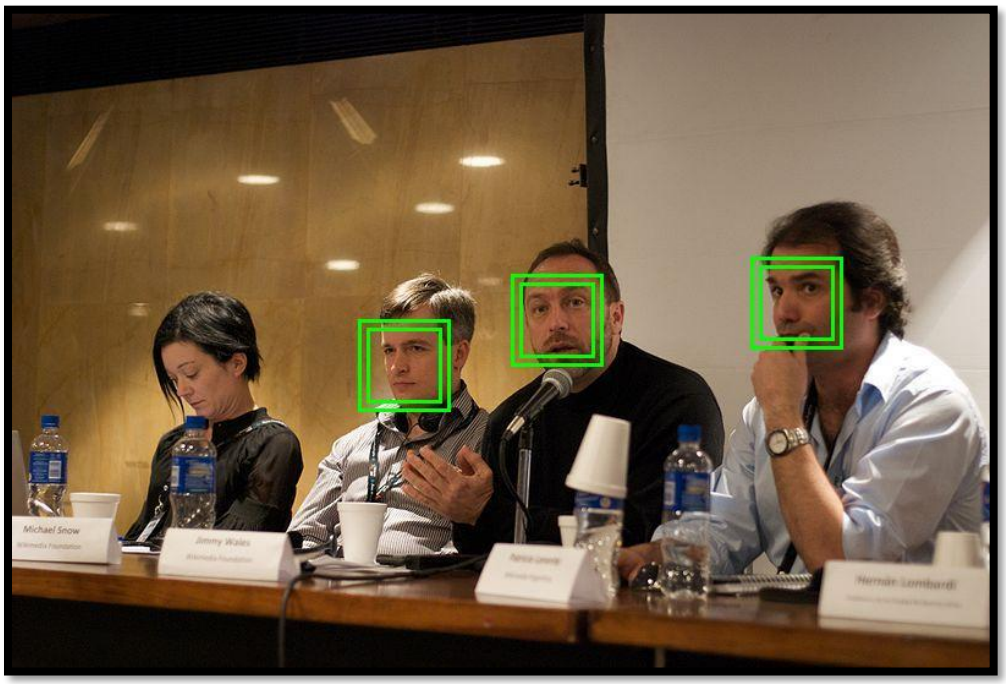


Figura 19. Reconocimiento facial utilizando OPENCV en PYTHON
Fuente: EE.UU. (2017); Sphinx

Tkinter

La siguiente información fue obtenida de: (Unipython, 2016). Primeramente, tenemos que conocer que son las interfaces graficas de usuario. Es una interfaz a través de la cual el usuario interactúa con dispositivos electrónicos como ordenadores, dispositivos portátiles y otros dispositivos. De igual forma son intermediarios entre el programa y el usuario. Está compuesta por iconos, menús, botones y cualquier representación gráfica.

Un lenguaje visual ha evolucionado a medida que la interfaz gráfica de usuario se ha convertido en algo común tanto en los sistemas operativos (SO) como en las aplicaciones de software. Incluso aquellos con pocos conocimientos de informática pueden ahora, a través del uso de GUI,

aprenden a utilizar aplicaciones informáticas para procesamiento de textos, finanzas, inventario, diseño, ilustraciones o pasatiempos.

Tkinter es una librería que proporciona a las aplicaciones de Python una interfaz de usuario fácil de programar. Además, es un conjunto de herramientas GUI de Tcl/Tk (Tcl: Tool Command Language), proporcionando una amplia gama de usos, incluyendo aplicaciones web, de escritorio, redes, administración, pruebas y muchos más.

Tkinter no es solo la única librería para Python especializada en la creación de interfaces gráficas, entre las más empleadas están wxPython, PyQt y PyGtk, todas con ventajas y desventajas. Entre los puntos fuertes que caracterizan a Tkinter en la creación de GUI, es que viene instalado con Python en casi todas las plataformas, su sintaxis es clara, fácil de aprender y documentación completa. (Véase la Figura N° 20)

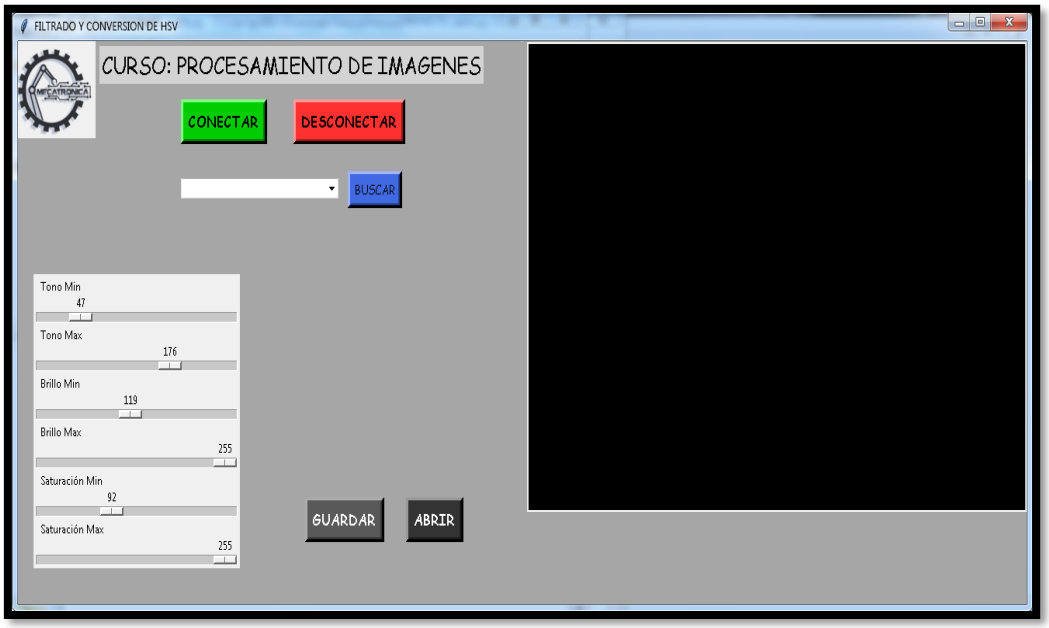


Figura 20. Interfaz gráfica de reconocimiento de color con TKINTER y OPENCV
Fuente: EE.UU. (2016); Unipython

2.3.4 Procesamiento de imágenes

La siguiente información fue obtenida de: (K., 2013). El procesamiento de imágenes tiene que ver con la adquisición, transmisión, procesamiento y representación de las imágenes. Las técnicas de proceso de imágenes se utilizan para mejorar la apariencia visual de las imágenes para un observador y para preparar convenientemente el contenido fotográfico de cara a la percepción por parte de máquinas. El proceso digital de imágenes se puede dividir en las siguientes áreas:

Adquisición o captura que se ocupa de los diferentes caminos para la obtención de imágenes; por ejemplo, utilizando cámaras digitales o digitalizando imágenes analógicas (fotografías).

Realce y mejora son las técnicas que se usan para mejorar la apariencia visual de las imágenes o para recuperar o restaurar las imágenes degradadas.

Segmentación que se ocupa de la división de las imágenes en regiones o áreas significativas.

Extracción de características que se ocupa de la detección y localización de entidades geométricas simples y complejas. Desde entidades simples como líneas y puntos hasta geometrías complejas como curvas y cuádricas.

El propósito de este tema es presentar los aspectos relevantes del procesamiento de imágenes, centrándose en torno a las herramientas y procesos del Realce y Mejora y la Extracción de características.

Realce y mejora de la imagen

El propósito de las técnicas de realce de la imagen es mejorar la apariencia de la misma para el observador. Es un proceso subjetivo, realizado habitualmente de una forma interactiva. La selección de los métodos

apropiados y la elección de los parámetros adecuados dependen de la calidad de la imagen original y de la aplicación.

Existe una gran cantidad de transformaciones u operaciones que se pueden realizar sobre las imágenes con el propósito de realzarlas y mejorarlas. Existen varios criterios para clasificar estas operaciones.

Transformación de Espacio Vectorial RGB a HSV

La siguiente información fue extraída de (EcuRed, 2018). Espacio de color HSV. Representación tridimensional del color basado en los componentes de tinte, matiz o tonalidad (hue, en inglés), saturación (saturation) y brillo o valor (value).

A diferencia del modelo RGB ampliamente usado en los monitores, televisores, etc., si bien las coordenadas de aquel son euclidianas; el color HSV sigue una representación más parecida a las coordenadas cilíndricas. Además, es una representación más cercana a la forma en que los humanos perciben los colores y sus propiedades, pues se agrupan las tonalidades de color, lo cual es distinto al caso RGB donde los colores no están necesariamente tan agrupados. (Véase la Figura N° 21)

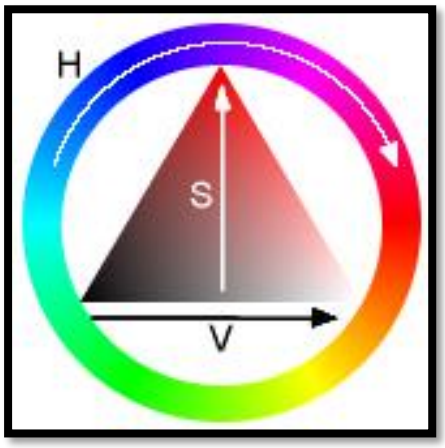


Figura 21. Espacio de color sobre las componentes de color: Tinte, saturación y brillo. Fuente: EE.UU. (2018); EcuRed

El modelo de color HSV es una transformación no lineal del modelo RGB en coordenadas cilíndricas de manera que cada color viene definido por las siguientes dimensiones:

Tinte o matiz: Se representa como un grado de ángulo cuyos valores posibles van de 0 a 360° (aunque para algunas aplicaciones se normalizan del 0 al 100%). Cada valor corresponde a un color. Ejemplos: 0 es rojo, 60 es amarillo y 120 es verde.

De forma intuitiva se puede realizar la siguiente transformación para conocer los valores básicos RGB:

Disponemos de 360 grados dónde se dividen los 3 colores RGB, eso da un total de 120º por color, sabiendo esto podemos recordar que el 0 es rojo RGB (1, 0, 0), 120 es verde RGB (0, 1, 0) y 240 es azul RGB (0, 0, 1). Para colores mixtos se utilizan los grados intermedios, el amarillo, RGB (1, 1, 0) está entre rojo y verde, por lo tanto 60º. Se puede observar cómo se sigue la secuencia de sumar 60 grados y añadir un 1 o quitar el anterior:

Saturación: Se representa como la distancia al eje de brillo negro-blanco. Los valores posibles van del 0 al 100%. A este parámetro también se le suele llamar "pureza" por la analogía con la pureza de excitación y la pureza colorimétrica de la colorimetría. Cuanto menor sea la saturación de un color, mayor tonalidad grisácea habrá y más decolorado estará. Por eso es útil definir la insaturación como la inversa cualitativa de la saturación. (Véase la Figura N° 22)

Figura 22. Graduaciones de saturación en el modelo HSV
Fuente: EE.UU. (2018); EcuRed

Valor: Representa la altura en el eje blanco-negro. Los valores posibles van del 0 al 100%. 0 siempre es negro. Dependiendo de la saturación, 100 podría ser blanco o un color más o menos saturado.

Transformación RGB a HSV: Sea MAX el valor máximo de los componentes (R, G, B), y MIN el valor mínimo de esos mismos valores, los componentes del espacio HSV se pueden calcular como puede observarse en la ecuación 12:

$$\boldsymbol{H} = \begin{cases} \boldsymbol{No\ definido}, & \boldsymbol{si\ MAX=MIN} \\ \boldsymbol{60^0 x \frac{G-B}{MAX-MIN} + 0^0}, & \boldsymbol{si\ MAX=R\ y\ G \geq B} \\ \boldsymbol{60^0 x \frac{G-B}{MAX-MIN} + 360^0}, & \boldsymbol{si\ MAX=R\ y\ G < B} \\ \boldsymbol{60^0 x \frac{B-R}{MAX-MIN} + 120^0}, & \boldsymbol{si\ MAX=G} \\ \boldsymbol{60^0 x \frac{R-G}{MAX-MIN} + 240^0}, & \boldsymbol{si\ Max= B} \end{cases} \quad \textbf{(12)}$$

$$S = \begin{cases} 0, & si\ MAX = 0 \\ 1 - \dfrac{MIN}{MAX}, & en\ otro\ caso \end{cases}$$

$$V = MAX$$

Transformaciones Morfológicas

La morfología matemática se basa en operaciones de teoría de conjuntos. En el caso de imágenes binarias, los conjuntos tratados son subconjuntos de Z^2 y en el de las imágenes en escala de grises, se trata de conjuntos de puntos con coordenadas en Z^3. Las operaciones morfológicas simplifican imágenes y conservan las principales características de forma de los objetos.

Un sistema de operadores de este tipo y su composición, permite que las formas subyacentes sean identificadas y reconstruidas de forma Morfología óptima a partir de sus formas distorsionadas y ruidosas. La morfología matemática se puede usar, entre otros, con los siguientes objetivos:

- Preprocesamiento de imágenes (supresión de ruidos, simplificación de formas).

- Destacar la estructura de los objetos (extraer el esqueleto, detección de objetos, envolvente convexa, ampliación, reducción,)
- Descripción de objetos (área, perímetro,)

Las transformaciones morfológicas son algunas operaciones simples basadas en la forma de la imagen. Normalmente se realiza en imágenes binarias. Necesita dos entradas, una es nuestra imagen original, la segunda se llama elemento de estructuración o kernel que decide la naturaleza de la operación. Dos operadores morfológicos básicos son la erosión y la dilatación. Luego, sus variantes de formas como Apertura, Cierre, Gradiente, etc. también entran en juego. Los veremos uno por uno con la ayuda de la siguiente imagen: (Véase la Figura N° 23)

Figura 23. Imagen de prueba
Fuente: EE.UU. (2013); Alexander Mordvintsev y Abid K

Erosión

La idea básica de la erosión es como la erosión del suelo solamente, erosiona los límites del objeto de primer plano (siempre trate de mantener el primer plano en blanco). Entonces ¿Qué es lo que hace? El kernel se desliza a través de la imagen (como en la convolución 2D). Un píxel en la imagen original (1 o 0) se considerará 1 solo si todos los píxeles debajo del kernel son 1, de lo contrario se erosionará (se pondrá a cero).

Entonces, lo que sucede es que todos los píxeles cerca del límite se descartarán dependiendo del tamaño del kernel. Por lo tanto, el grosor o el tamaño del objeto en primer plano disminuye o simplemente la región

blanca disminuye en la imagen. Es útil para eliminar pequeños ruidos blancos (como hemos visto en el capítulo del espacio de color), separar dos objetos conectados, etc. (Véase la Figura N° 24)

Figura 24. Imagen con filtro de erosión.
Fuente: EE.UU. (2013); Alexander Mordvintsev y Abid K

Dilatación

Es justo lo contrario de la erosión. Aquí, un elemento de píxel es '1' si al menos un píxel debajo del núcleo es '1'. Por lo tanto, aumenta la región blanca en la imagen o el tamaño del objeto en primer plano aumenta. Normalmente, en casos como la eliminación de ruido, la erosión es seguida por la dilatación. Porque, la erosión elimina los ruidos blancos, pero también encoge nuestro objeto. Así lo dilataremos. Como el ruido se ha ido, no volverán, pero nuestra área de objetos aumenta. También es útil para unir partes rotas de un objeto. (Véase la Figura N° 25)

Figura 25. Imagen con filtro de dilatación.
Fuente: EE.UU. (2013); Alexander Mordvintsev y Abid K

Gradiente

Esta operación es equivalente a la diferencia entre las operaciones de dilatación y erosión, esta operación es bastante útil cunado deseamos buscar la silueta de una figura. (Véase la Figura N° 26)

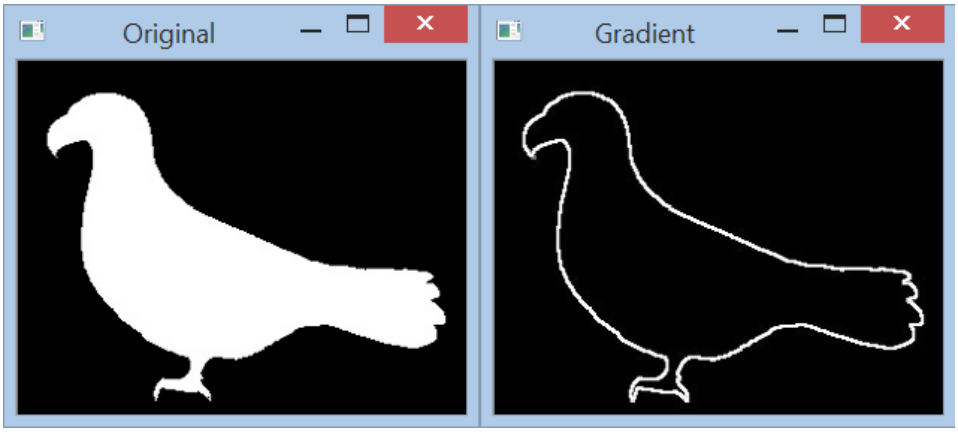


Figura 26. Imagen original e imagen con gradiente.
Fuente: EE.UU. (2013); Alexander Mordvintsev y Abid K

Apertura

Apertura es solo otro nombre de erosión seguido de dilatación. Es útil para eliminar el ruido. (Véase la Figura N° 27)

Figura 27. Eliminación de ruido mediante filtro de apertura.
Fuente: EE.UU. (2013); Alexander Mordvintsev y Abid K

Cierre

El cierre es al revés de Apertura, Dilatación seguida de Erosión. Es útil para cerrar pequeños orificios dentro de los objetos en primer plano o pequeños puntos negros en el objeto. (Véase la Figura N° 28)

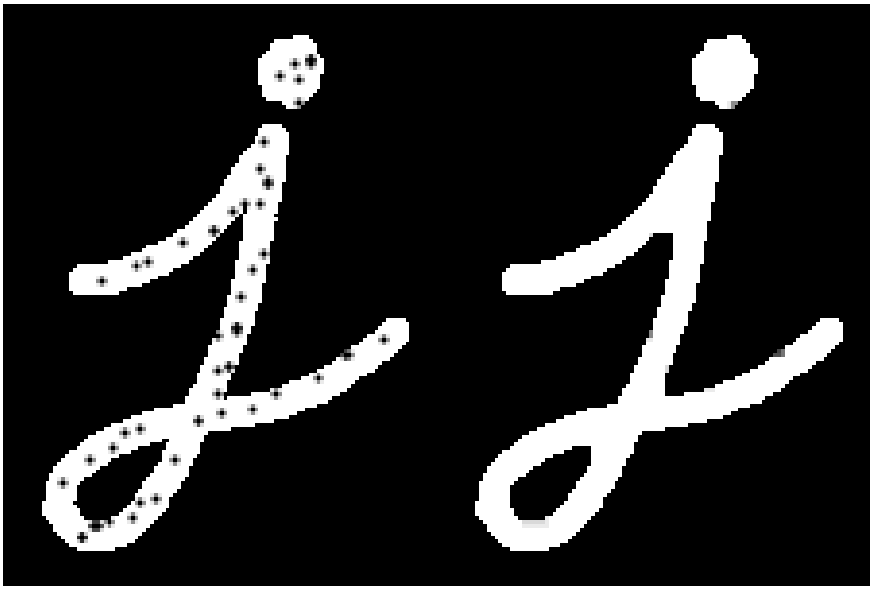


Figura 28. Aplicación de filtro de cierre para completar espacios en la imagen. Fuente: EE.UU. (2013); Alexander Mordvintsev y Abid K

2.3.5 Neurona Perceptrón

La siguiente información fue obtenida de (Valdivieso, 2017). Las Redes Neuronales (NN: Neural Networks) fueron originalmente una simulación abstracta de los sistemas nerviosos biológicos, constituidos por un conjunto de unidades llamadas neuronas o nodos conectados unos con otros.

El primer modelo de red neuronal fue propuesto en 1943 por McCulloch y Pitts en términos de un modelo computacional de actividad nerviosa. Este modelo binario, donde cada neurona tenía un escalón o umbral prefijado, y sirvió de base para los modelos posteriores.

Una primera clasificación de los modelos de NN es:

1. **Modelos inspirados en la Biología:** Estos comprenden las redes que tratan de simular los sistemas neuronales biológicos, así como ciertas funciones como las auditivas o de visión.
2. **Modelos artificiales aplicados:** Estos modelos no tienen por qué guardar similitud estricta con los sistemas biológicos. Sus

arquitecturas están bastante ligadas a las necesidades de las aplicaciones para las que son diseñados.

Redes Neuronales de tipo biológico

Se estima que el cerebro humano contiene más de cien mil millones (10^{11}) de neuronas y 10^{14} sinapsis en el sistema nervioso. Los estudios realizados sobre la anatomía del cerebro humano concluyen que hay, en general, más de 1000 sinapsis por término medio a la entrada y a la salida de cada neurona.

Aunque el tiempo de conmutación de las neuronas biológicas (unos pocos milisegundos) es casi un millón de veces mayor que en las actuales componentes de las computadoras, las neuronas naturales tienen una conectividad miles de veces superior a la de las artificiales.

El objetivo principal de las redes de tipo biológico es desarrollar operaciones de síntesis y procesamiento de información, relacionadas con los sistemas biológicos.

La neurona artificial o unidad básica de inferencia en forma de discriminador lineal, a partir de lo cual se desarrolla un algoritmo capaz de generar un criterio para seleccionar un subgrupo a partir de un grupo de componentes más grande.

La limitación de este algoritmo es que, si dibujamos en un gráfico estos elementos, se deben poder separar con un hiperplano únicamente los elementos "deseados" discriminándolos (separándolos) de los "no deseados".

El perceptrón puede utilizarse con otros tipos de perceptrones o de neurona artificial, para formar una red neuronal artificial más compleja. (Véase la Figura N° 29).

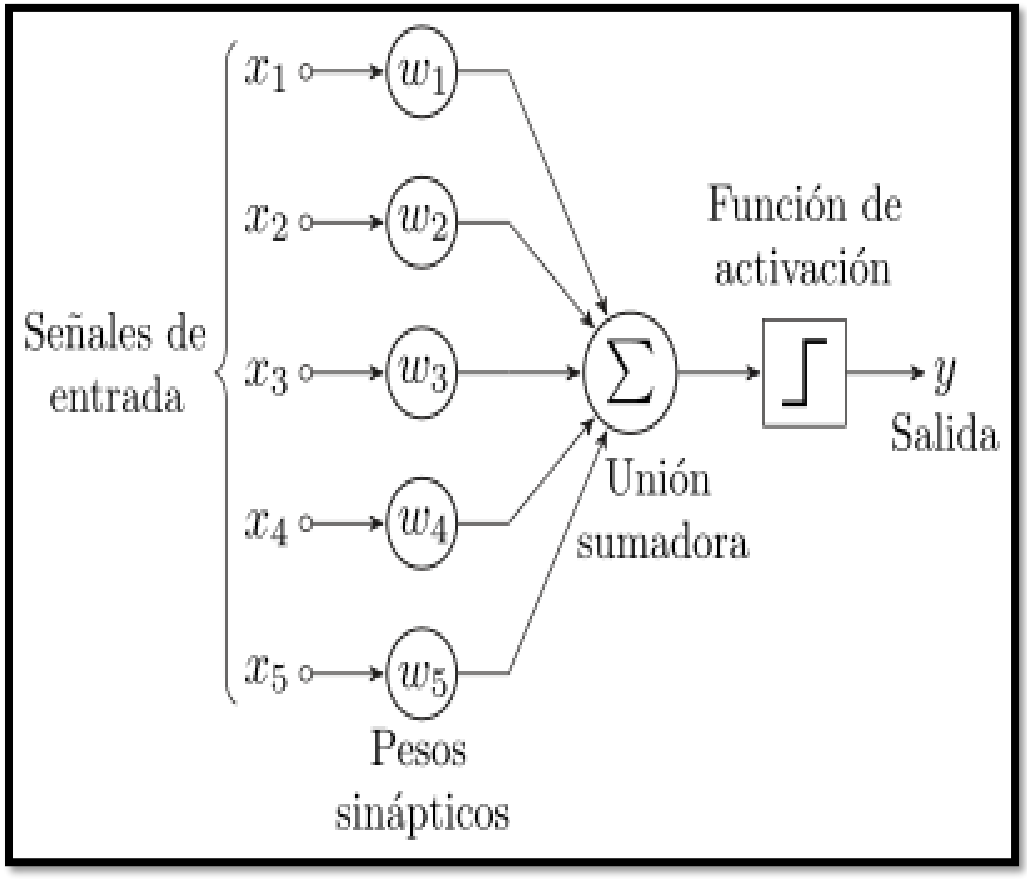


Figura 29. Diagrama de un perceptrón con cinco señales de entrada
Fuente: México (2017); Valdivieso, Pedro A Castillo

Definición de función de activación

La función de activación se encarga de devolver una salida a partir de un valor de entrada, normalmente el conjunto de valores de salida en un rango determinado como (0,1) o (-1,1).

Se buscan funciones que las derivadas sean simples, para minimizar con ello el coste computacional.

Tipos de función de activación

Sigmoid – Sigmoide

La función sigmoide transforma los valores introducidos a una escala (0,1), donde los valores altos tienen de manera asintótica a 1 y los valores muy

bajos tienden de manera asintótica a 0. Como puede observarse en la ecuación 13

$$f_{(x)} = \frac{1}{1+e^{-x}} \qquad \textbf{(13)}$$

Características de la función sigmoide (Véase la Figura N° 30):

- Satura y mata el gradiente.
- Lenta convergencia.
- No está centrada en el cero.
- Está acotada entre 0 y 1.
- Buen rendimiento en la última capa.

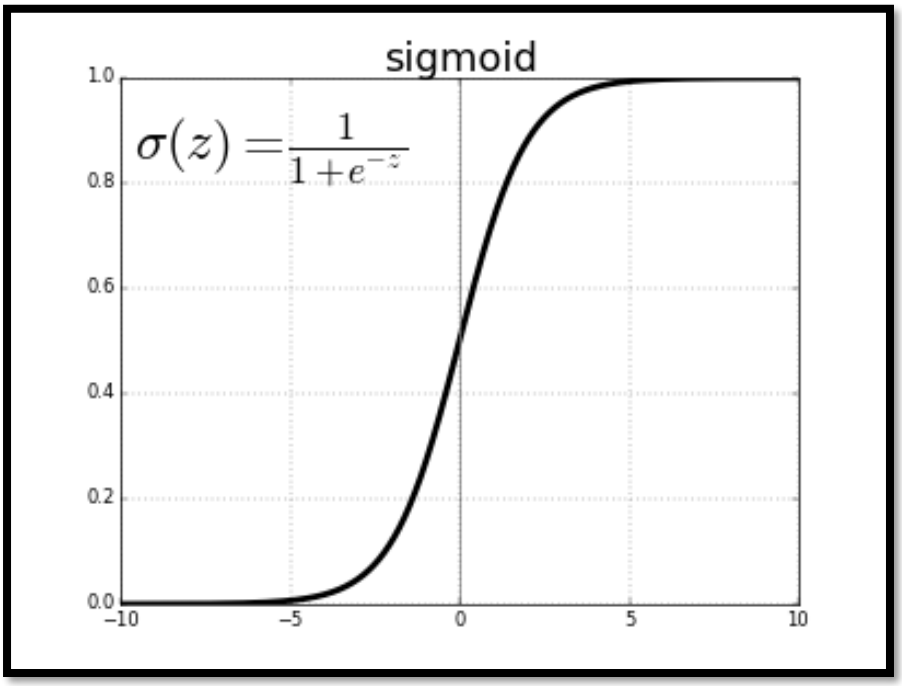


Figura 30. Función de activación: Sigmoide
Fuente: México (2017); Valdivieso, Pedro A Castillo

Tanh – Tangent Hyperbolic – Tangente hiperbólica

La función tangente hiperbólica transforma los valores introducidos a una escala (-1,1), donde los valores altos tienen de manera asintótica a 1 y los valores muy bajos tienden de manera asintótica a -1. Como puede verse en la ecuación 14.

$$f_{(x)} = \frac{2}{1+e^{-2x}} - 1 \qquad \textbf{(14)}$$

Características de la función tangente hiperbólica (Véase la Figura N° 31):

- Muy similar a la sigmoide
- Satura y mata el gradiente.
- Lenta convergencia.
- Centrada en 0.
- Está acotada entre -1 y 1.
- Se utiliza para decidir entre uno opción y la contraria.
- Buen desempeño en redes recurrentes.

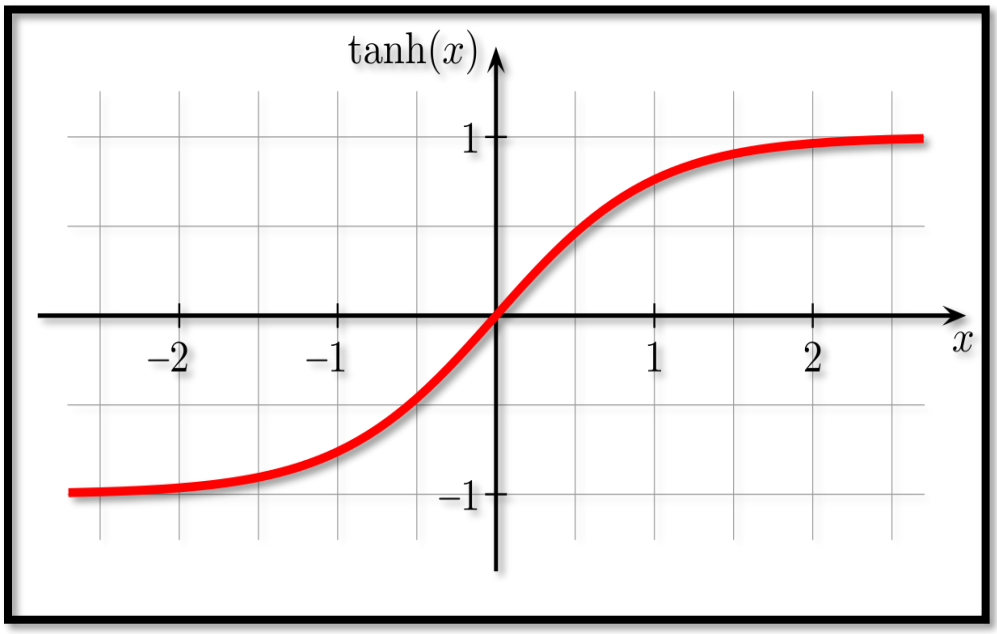


Figura 31. Función de activación: Tangente hiperbólica
Fuente: México (2017); Valdivieso, Pedro A Castillo

ReLU – Rectified Lineal Unit

La función ReLU transforma los valores introducidos anulando los valores negativos y dejando los positivos tal y como entran. Como puede verse en la ecuación 15

$$f_{(x)} = \max(0, x) = \begin{cases} 0\ si: x < 0 \\ x\ si: x \geq 0 \end{cases} \quad \textbf{(15)}$$

Características de la función ReLU (Véase la Figura N° 32):

- Activación Sparse – solo se activa si son positivos.
- No está acotada.
- Se pueden morir demasiadas neuronas.
- Se comporta bien con imágenes.
- Buen desempeño en redes convolucionales.

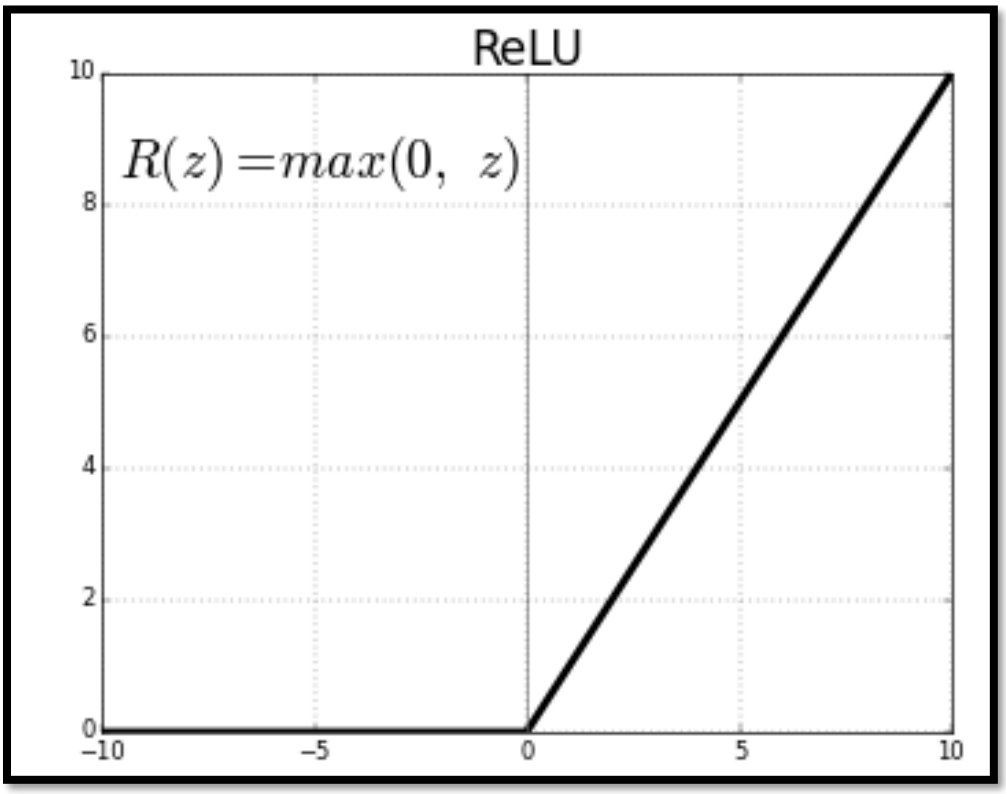


Figura 32. Función de activación: ReLU
Fuente: México (2017); Valdivieso, Pedro A Castillo

Función Escalón

Transforma los valores introducidos anulando los valores negativos y transformando en 1 los valores positivos como puede verse en la ecuación 16

$$f_{(x)} = \max(0,1) = \begin{cases} 0 \; si \; x < 0 \\ 1 \; si \; x \geq 0 \end{cases} \quad \textbf{(16)}$$

Características de la función Escalón (Véase la Figura N° 33):

- Activación Sparse – solo se activa si son positivos.
- No está acotada.
- Se pueden morir demasiadas neuronas.
- Se comporta bien con imágenes.
- Buen desempeño en redes convolucionales.

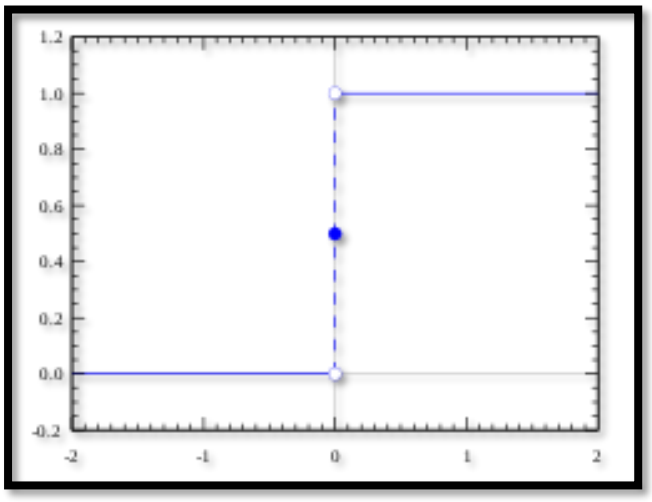


Figura 33. Función de activación: Escalón
Fuente: México (2017); Valdivieso, Pedro A Castillo

Algoritmo de Corrección de Error

La siguiente información fue obtenida de ("Red Neuronal de Topología Flexible", 1999).Los algoritmos de corrección de error están en la categoría de los algoritmos con aprendizaje supervisado, se caracterizan por el cambio de los pesos de la neurona después de una presentación de un patrón para corregir el error en la salida, a su vez se clasifican en dos grupos: α-LMS y el Perceptron.

α – LMS (Least – Mean- Square)

En el análisis de la regla de aprendizaje α-LMS, la neurona utiliza la función de activación lineal “purelin”, donde la activación es igual a la salida de la red ($S_k = Y_k$), (Véase la Figura N° 34).

El vector de pesos actual w^k se modifica en la dirección del patrón de entrada x^k para producir un nuevo vector de pesos w^{k+1}. La actualización del vector de pesos esta dada por la ecuación 17.

$$w^{k+1} = w^k + \alpha e^k \frac{x^k}{\|x^k\|^2} \quad k = 1,2, \dots iteraciones \qquad \textbf{(17)}$$

Donde, w^k es el vector de pesos, x^k es el patrón de entrada, w^{k+1} es el nuevo vector de pesos, α es el factor de aprendizaje (determina la aceleración de convergencia), e^k es el error, $\|x\|$ es la norma del vector y $\|x^k\|^2$ es igual al número de pesos. La regla de aprendizaje de (17) es derivada de la regla de Widrow-Hoff

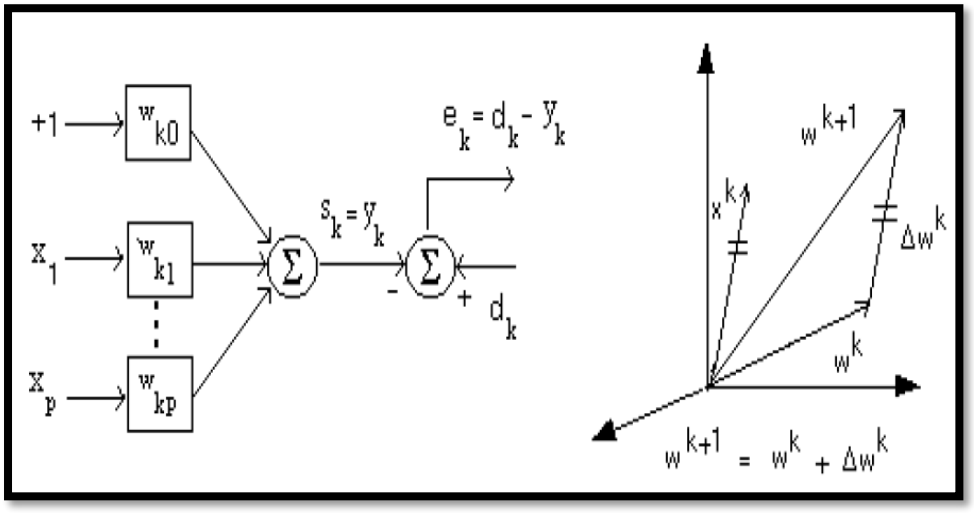


Figura 34. Combinador lineal adaptativo - Actualizando el vector de pesos
Fuente: La Paz (1999); Aguilar, R.

La norma se presenta: Como puede verse en la ecuación 18.

$$\|x\| = \sqrt{x_1^2 + x_2^2 + \dots + x_n^2} \qquad \textbf{(18)}$$

El vector normalizado podemos verlo en la ecuación 19.

$$x = [\frac{x_1}{\|x\|} \quad \frac{x_2}{\|x\|} \quad \ldots \quad \frac{x_n}{\|x\|}]^T \qquad \textbf{(19)}$$

Analizamos la ecuación (17) para calcular el error en la iteración k-ésima. Utilizamos la ecuación 20.

$$e^k = d^k - s^k \qquad \textbf{(20)}$$

En la figura N° 33 se muestra que la salida del sumador es como se muestra en la ecuación 21.

$$s = (w^k)^T x^k \qquad \textbf{(21)}$$

Reemplazando la ecuación 21 en la 20: Obtenemos la ecuación 22.

$$e^k = d^k - (w^k)^T x^k \qquad \textbf{(22)}$$

El cambio del error con respecto al vector de pesos esa dado por la ecuación 23

$$\Delta e^k = -(x^k)^T \Delta w^k \qquad \textbf{(23)}$$

Reemplazamos la ecuación 7 en la ecuación de la resultante de los vectores de pesos de la figura N° 33, obtenemos como resultado la ecuación 24.

$$\Delta w^k = \ w^{k+1} - w^k = w^k + \alpha e^k \frac{x^k}{\|x^k\|^2} - w^k = \alpha e^k \frac{x^k}{\|x^k\|^2} \qquad \textbf{(24)}$$

Sustituimos la ecuación 24 en 23 y consideramos $((x^k)^T x^k = \|x^k\|^2)$ y obtenemos la ecuación 25.

$$\Delta e^k = -\alpha e^k \qquad \textbf{(25)}$$

La velocidad de convergencia del algoritmo α-LMS depende de la correlación existente entre los patrones de entrada, a mayor correlación menor velocidad. La correlación al igual que la Convolución, la correlación es una operación entre dos secuencias. Pero al contrario de la Convolución, el objetivo de la correlación es medir el parecido entre dos señales y así extraer información. Cuando los patrones están fuertemente

correlacionados, el combinador lineal maneja información redundante y su aprendizaje es lento. El parámetro a determina el decrecimiento del error durante el entrenamiento en la práctica, 0.1 <α < 1.

Perceptrón con Momentum

La siguiente información fue obtenida de ("Red Neuronal de Topología Flexible", 1999). El perceptrón modificado (con momentum), es una versión mejorada del perceptrón aprende la regla. Estas redes generalmente usan una función de activación limitador duro (véase la Figura N°35), es decir, los datos que manejan son bipolares debido a que el aprendizaje del entrenamiento es bastante rápido, lo que resulta un cómputo beneficioso.

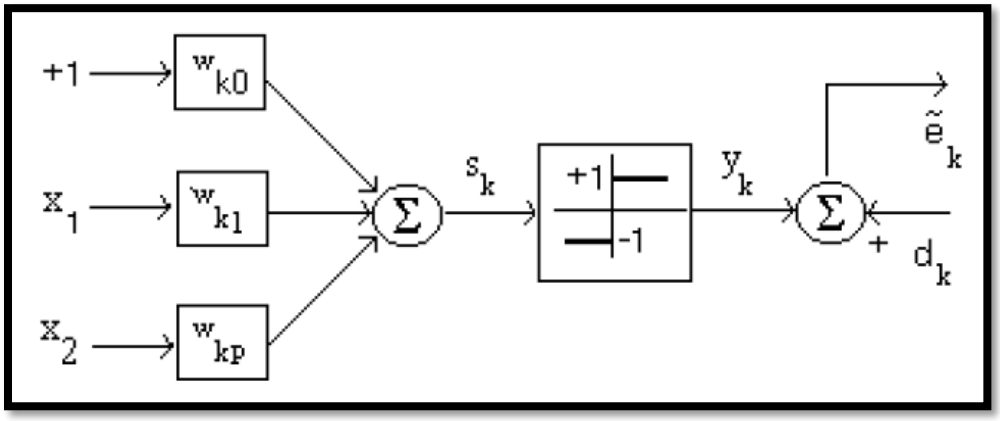


Figura 35. Neurona artificial con función de activación limitador duro.
Fuente: La Paz (1999); Aguilar, R.

El error no lineal durante la presentación del patrón k-ésima obedece a la ecuación 26 que se muestra a continuación.

$$e^{\sim k} = d^k - y^k \quad \textbf{(26)}$$

La salida de la red es. Como puede verse en la ecuación 27

$$y^k = \sigma(s^k) \quad \textbf{(27)}$$

La actualización del vector de pesos se hace con la regla del perceptrón. Como puede verse en la ecuación 28

$$w^{k+1} = w^k + \alpha e^{\sim k} x^k \quad \textbf{(28)}$$

Este algoritmo es válido solamente para respuestas deseadas bipolares {-1,1} debido a esto utilizan una función de activación limitador duro. Para las aplicaciones, los patrones deben ser linealmente separables o de lo contrario el algoritmo oscilará. La versión mejorada de la regla del perceptrón viene dada por la ecuación 29.

$$w^{k+1} = w^k + \alpha e^{\sim k} x^k + \mu(w^k - w^{k-1}) \quad \textbf{(29)}$$

Siendo el término adicional denominado momentum (μ), es una memoria o inercia que permite que los cambios en el vector de pesos w sean suaves porque incluyen información sobre el cambio anterior. En la práctica este valor está comprendido 0.1 < μ < 1.0.

Aprendizaje

En el perceptrón, existen dos tipos de aprendizaje, el primero utiliza una tasa de aprendizaje mientras que el segundo no la utiliza. Esta tasa de aprendizaje amortigua el cambio de los valores de los pesos.

El algoritmo de aprendizaje es el mismo para todas las neuronas, todo lo que sigue se aplica a una sola neurona en el aislamiento. Se definen algunas variables primero:

- x(j) denota el elemento en la posición j en el vector de la entrada.
- W(j) el elemento en la posición j en el vector de peso.
- y denota la salida de la neurona.
- δ denota la salida esperada.
- α es una constante tal que 0< α <1.

Los dos tipos de aprendizaje difieren en este paso. Para el primer tipo de aprendizaje, utilizando tasa de aprendizaje, utilizaremos la siguiente regla de actualización de los pesos como puede verse en la ecuación 30:

$$w(j)' = w(j) + \alpha(\delta - y)x(j) \quad \textbf{(30)}$$

Para el segundo tipo de aprendizaje, sin utilizar tasa de aprendizaje, la regla de actualización de los pesos será como podemos verlo en la ecuación 31:

$$w(j)' = w(j) + (\delta - y)x(j) \quad \textbf{(31)}$$

Por lo cual, el aprendizaje es modelado como la actualización del vector de peso después de cada iteración, lo cual sólo tendrá lugar si la salida y difiere de la salida deseada δ . Para considerar una neurona al interactuar en múltiples iteraciones debemos definir algunas variables más:

- x_i denota el vector de entrada para la iteración i
- w_i denota el vector de peso para la iteración i
- y_i denota la salida para la iteración i
- $D_m = \{(x_1, y_1), \dots, (x_m, y_m)\}$ denota un periodo de aprendizaje de m iteraciones

En cada iteración el vector de peso es actualizado como sigue:

Para cada pareja ordenada (x, y) en $D_m = \{(x_1, y_1), \dots, (x_m, y_m)\}$

Pasar (x_i, y_i, w_i) a la regla de actualización (véase la Ecuación N° 20)

El periodo de aprendizaje D_m se dice que es separable linealmente si existe un valor positivo δ y un vector de peso w tal que: podemos ver en la ecuación 32 para todos los i.

$$y_i.(< w, x_i > + u) > \gamma \quad \textbf{(32)}$$

Novikoff (1962) probó que el algoritmo de un número finito de iteraciones si los datos son separables linealmente y el número de errores está limitado como puede verse en la ecuación 33.

$$(\frac{2R}{\gamma})^2 \quad \textbf{(33)}$$

Sin embargo, si los datos no son separables linealmente, la línea de algoritmo anterior no se garantiza que converja.

Ventajas

Las redes neuronales artificiales (RNA) tienen muchas ventajas debido a que están basadas en la estructura del sistema nervioso, principalmente el cerebro.

Aprendizaje: Las RNA tienen la habilidad de aprender mediante una etapa que se llama etapa de aprendizaje. Esta consiste en proporcionar a la RNA datos como entrada a su vez que se le indica cuál es la salida (respuesta) esperada.

Auto organización: Una RNA crea su propia representación de la información en su interior, descargando al usuario de esto.

Tolerancia a fallos: Debido a que una RNA almacena la información de forma redundante, ésta puede seguir respondiendo de manera aceptable aun si se daña parcialmente.

Flexibilidad: Una RNA puede manejar cambios no importantes en la información de entrada, como señales con ruido u otros cambios en la entrada (ej. si la información de entrada es la imagen de un objeto, la respuesta correspondiente no sufre cambios si la imagen cambia un poco su brillo o el objeto cambia ligeramente)

Tiempo real: La estructura de una RNA es paralela, por lo cual, si esto es implementado con computadoras o en dispositivos electrónicos especiales, se pueden obtener respuestas en tiempo real.

Desventajas

- Complejidad de aprendizaje para grandes tareas, cuantas más cosas se necesiten que aprenda una red, más complicado será enseñarle.
- Tiempo de aprendizaje elevado. Esto depende de dos factores: primero si se incrementa la cantidad de patrones a identificar o clasificar y segundo si se requiere mayor flexibilidad o capacidad de adaptación de la red neuronal para reconocer patrones que sean

sumamente parecidos, se deberá invertir más tiempo en lograr que la red converja a valores de pesos que representen lo que se quiera enseñar.

2.4 Definición de términos básicos

Circuito Adaptador de Niveles de Tensión

Circuito electrónico que permite adaptar los niveles de tensión de una balanza (0-10mV) con los del Arduino (0-5V). Utilizando el CHIP HX710B Convertidor Analógico Digital con una resolución de 24 bits. Para obtener uno de los parámetros de entrada de la Neurona Perceptrón

Algoritmo de Reconocimiento de Colores

El "reconocimiento de color" es la capacidad de un sensor o algoritmo de distinguir colores a partir de la extracción de información de la luz. La manera básica de detectar el color consiste en captar la luz incidente en un sensor (cámara). Esto, mediante un conjunto de celdas de fotones que forman una matriz de puntos, uno por cada pixel, es capaz de medir la cantidad de luz llegada a cada uno de estos, produciendo una corriente eléctrica que varía en función de la intensidad de luz recibida. Una vez se ha medido la cantidad de luz se procede a la detección de colores. La técnica que utilizaremos para el reconocimiento de colores es la transformación de espacio vectorial RGB a HSV ya que es un algoritmo adecuado para hacer seguimiento a una imagen por su color, luego aplicaremos unos filtros morfológicos de erosión y dilatación para eliminar el ruido presente en la imagen, luego mediante capas separaremos el color rojo amarillento del verde amarillento y luego calcularemos sus áreas quien tenga mayor área será el color predominante del mango.

Trackbar’s

El algoritmo de reconocimiento de color requiere ser calibrado antes de ser utilizado, para lo cual utilizaremos la técnica de transformación de espacio

vectorial RGB a HSV, por lo cual tendremos que manipular 6 variables (Hmin, Hmax, Smin, Smax, Vmin y Vmax) para eso utilizaremos 6 trackbar's que varían desde (0-255)

Comunicación Serial (RS-232)

RS-232 (Recommended Standard 232, en español: "Estándar Recomendado 232"), también conocido como EIA/TIA RS-232C, es una interfaz que designa una norma para el intercambio de datos binarios serie entre un DTE (Data Terminal Equipment, "Equipo Terminal de Datos"), como por ejemplo una computadora, y un DCE (Data Communication Equipment, "Equipo de Comunicación de Datos"), por ejemplo, un módem. Existen otros casos en los que también se utiliza la interfaz RS-232. Una definición equivalente publicada por la UIT se denomina V.24.

En particular, existen ocasiones en que interesa conectar otro tipo de equipamientos, como pueden ser computadoras. Evidentemente, en el caso de interconexión entre los mismos, se requerirá la conexión de un DTE con otro DTE. Para ello se utiliza una conexión entre los dos DTE sin usar módem, por ello se llama módem nulo (null modem).

III HIPÓTESIS Y VARIABLES

3.1 Hipótesis

3.1.1 Hipótesis general

“El desarrollo de un algoritmo basado en la teoría de redes neuronales cuyos parámetros de entrada sean el color (procesamiento de imágenes) y el peso (extracción paralela los valores de una balanza) permitirá la automatización del proceso de selección de mangos”

3.1.2 Hipótesis especificas

Hipótesis Específica N°1: “Es factible la implementación de un circuito electrónico utilizando el Módulo HX710B y el Arduino para extraer los valores de la Balanza, para luego importarlos a Python”.

Hipótesis Específica N°2: “Es viable la implementación de un prototipo de hardware a escala para la captura de imágenes con la finalidad de mantener constante el ambiente”.

Hipótesis Específica N°3: “Es viable el desarrollo de un código de programación en Python utilizando técnicas de reconocimiento de imágenes y técnicas de detección de bordes para reconocer el color de un mango”.

Hipótesis Específica N°4: “Es factible crear una Neurona Perceptrón en Python para clasificar los mangos en 2 tipos (Local y Extranjero) cuyos valores de entrada de la neurona sean el color y el peso”.

3.2 Definición Conceptual de variables

- **Variables Independientes**
- ✓ Algoritmo de Control Basado en Procesamiento de Imágenes y Redes Neuronales.

Se desarrolló un programa utilizando Python y las librerías de OpenCV (Procesamiento de Imágenes) y Tkinter (Interfaz Gráfica), el programa

cuenta con un Textbox, para seleccionar el puerto del Arduino, 2 botones uno para “Conectar” y “Desconectar” la webcam, también el programa cuenta con un selector de 3 opciones (Color1, Color2, Color3) debido a que los mangos tienen un poco de color (Rojo, Amarillo y Verde), cada una de las 3 opciones cuenta con 6 trackbar’s (Hmin, Hmax, Smin, Smax, Vmin y Vmax) adicionalmente el sistema cuenta con un botón de guardar y abrir, para no tener que realizar la calibración del sistema todos los días, como en todo sistema basado en técnicas de procesamiento de imágenes este está expuesto a ruido, para lo cual se le aplico técnicas de procesos morfológicos (Erosión y Dilatación), para saber cuál es el color que predomina en el mango se le calculo su área tanto del rojo, amarillo y verde y dependiendo de quien tenga mayor se emite una decisión, adicionalmente el sistema cuenta con una neurona perceptrón que permite clasificar la fruta en 2 grupos: Exportación y Consumo Local (Véase la figura N°36)

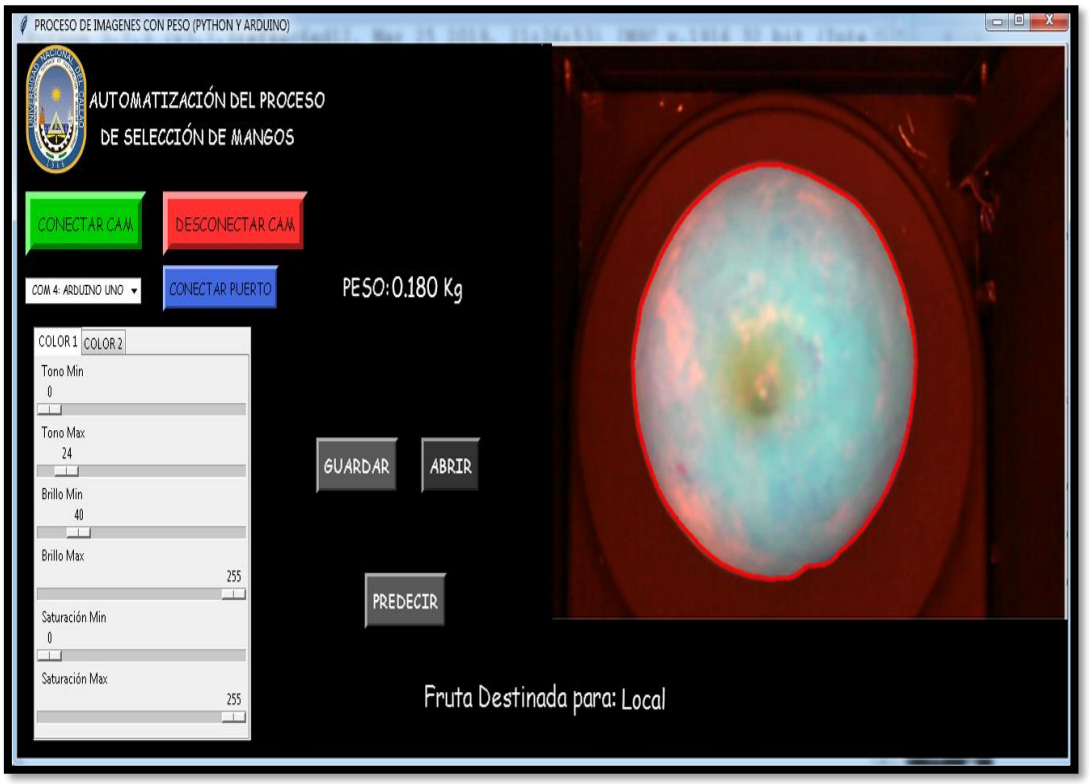


Figura 36. Programa de automatización de selección de mangos.
Fuente: UNAC (2019); Autoría propia.

- **Variables Dependientes**
- ✓ El Proceso de Selección de Mangos.

Es el error humano que existe al momento de clasificar los mangos en dos grupos: Consumo Local y los de Exportación, antes de la implementación del sistema de automatización de clasificación de mangos. Ya que antes la única forma de clasificar los mangos era mediante el criterio de los trabajadores, la característica de los mangos de exportación es de color verde-amarillento y son grandes ya que mientras viajan estos van madurando y los magos de consumo local son de color rojizo- amarillento y son de tamaño más pequeño en referencia con los de exportación. (Véase la figura N°37)

Figura 37. Proceso de selección de mangos.
Fuente: UNAC (2019); Autoría propia

3.2.1 Operacionalización de variables

TABLA 2. *OPERACIONALIZACIÓN DE LAS VARIABLES*

Variable	Dimensiones	Indicadores	Instrumento
X= Algoritmo de Control Basado en Procesamiento de Imágenes y Redes Neuronales	Peso y Color	Algoritmo de control diseñado aplicando Transformación de espacio vectorial RGB a HSV y procesos morfológicos hechos en Python para automatizar el proceso de selección de mangos.	Cámara y Balanza
Y= El Proceso de Selección de Mangos.	Mango Local y Extranjero	Proceso actual realizado por trabajadores que presenta varios errores	La vista y el tacto del trabajador

Fuente: UNAC (2019); Autoría propia.

IV DISEÑO METODOLOGICO

4.1 Tipo y diseño de investigación

4.1.1 Tipo de Investigación

- **Analítico**

Para realizar un análisis de los elementos que conforman este proyecto el método de investigación a utilizar es el Método Analítico, para así desintegrar, descomponer un todo (el algoritmo de control), en sus partes para estudiar cada uno de sus elementos, así como las relaciones entre sí con todo.

- **Experimental**

Porque se realizaron diferentes pruebas preliminares que nos permitirá desarrollar un algoritmo de procesamiento de imágenes, un circuito electrónico que permite extraer datos de manera paralela de una balanza y un algoritmo de redes neuronales que en conjunto nos permite la automatización del proceso de selección de mangos y pruebas finales que nos permitirá su performance.

- **Temporal**

Porque el estudio está circunscrito a un intervalo de tiempo Enero 2019-Agosto 2020

- **Espacial**

Se realizará la investigación tomando como referencia la huerta de mangos del Sr. Javier Carrasco Jimenez, ubicado en el distrito de Tambo grande, provincia de Piura departamento de Piura

4.1.2 Diseño de la Investigación

- **Extracción de Datos de una balanza**

Para la presente investigación utilizaremos una balanza casera modelo (SF-400) cuya capacidad es de 5 kg. (Véase la figura N°38). La razón por la cual utilizamos esta balanza es por qué cuenta con una celda de carga interna que ya viene con una estructura puesta lo cual facilita mucho el trabajo.

Figura 38. Balanza de 10 kg. Modelo SF-400
Fuente: UNAC (2019); Autoría Propia.

La celda de carga debe ser alimentada con 5v y tiene una salida de 0-10mv es decir que 5kg equivale a 10mv y para poder detectar 1gr es necesario encontrar un hardware que reconozca un voltaje de 2 microvoltios. Como puede verse en la ecuación 34:

$$V_{min} = \frac{5000gr}{0.01V} x1gr = 2\mu V \quad \textbf{(34)}$$

El Arduino UNO viene con microcontrolador ATmega328P que contiene un convertidor ADC de 10bits y el mínimo voltaje que reconoce es aproximadamente 4.8mV. Como puede verse en la ecuación 35.

$$V_{min} = \frac{5V}{1023bits} * 1bit \approx 4.8mV \quad \textbf{(35)}$$

Lo cual quiere decir que recién cuando ponga en la balanza 2.5kg el ADC del ARDUINO lo reconocerá como 1 bit y que cuando ponga recién 5Kg el ADC lo reconocerá como 2bits. Lo cual es insuficiente para nuestro caso puesto que normalmente un mango pesa entre 300gr y 700gr, por lo cual utilizaremos un CHIP HX710B, el cual posee un ADC de 24bits. (Véase la Figura N° 39)

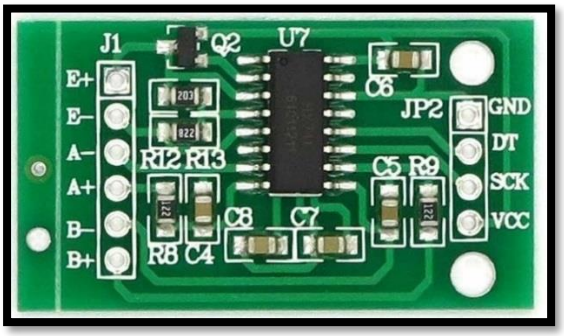


Figura 39. Chip HX710B, ADC de 24 bits
Fuente: UNAC (2019); Autoría Propia.

El CHIP HX710B es un chip especialmente diseñado para realizar comunicaciones entre las celdas de carga y los microcontroladores, este chip se acoplara de forma paralela a la celda de carga interna de la balanza. (Véase la Tabla N° 3)

TABLA 3. *TABLA DE CONEXIONES DEL CHIP HX7710B*

Pin	HX710B
E+	Cable Rojo de la celda de carga
E-	Cable Negro de la celda de carga
A-	Cable Verde de la celda de carga
A+	Cable Blanco de la celda de carga
DT	La salida de datos de 24 bits
SCK	Entrada de señal de reloj

Fuente: UNAC (2019); Autoría Propia.

Como ya explicamos anteriormente la salida de voltaje de la celda de carga es de 0- 10mV. Lo cual es un voltaje muy pequeño y muy sensible al ruido al ruido eléctrico que existe en el cable y al movimiento, para lo cual nosotros utilizamos un núcleo de ferrita (Véase la Figura N° 40) y el cable

de conexión entre la celda de carga y el HX710B debe ser lo más corto posible y los cables deben estar recubierto con termo contraíble

Figura 40. Núcleo de ferrita
Fuente: UNAC (2019); Autoría Propia.

El núcleo de ferrita, es un tipo de dispositivo diseñado para el filtro de corrientes “parasitas” que pasan a través de un conductor eléctrico.

Adicionalmente utilizamos un módulo LCD de 20x4 para poder visualizar el peso y poder verificar que el peso que aparece en la pantalla es el mismo que aparece en la balanza. (Véase la Figura N°41)

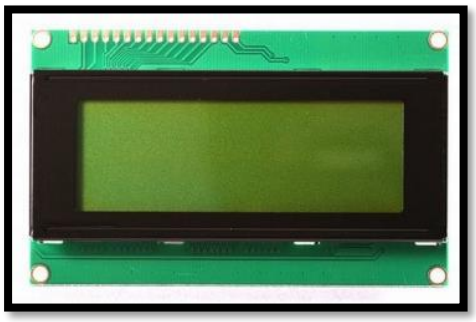


Figura 41. Pantalla LCD 20X4
Fuente: UNAC (2019); Autoría Propia.

La pantalla LCD va acompañada con un módulo I2C, la celda de carga va conectada con el módulo HX710B (ADC de 24 bits), luego este el módulo se conecta con el Arduino, el cual debe generar una señal de reloj de 1Khz para que el ADC le pueda entregar el dato del peso en una resolución de 24 bits, luego el Arduino se conecta con el módulo i2c y este a su vez se

conecta con el LCD de 20x4, a la par el Arduino transmite de manera serial el dato del peso a la interfaz gráfica desarrollada en Python. (Véase la Figura N° 42).

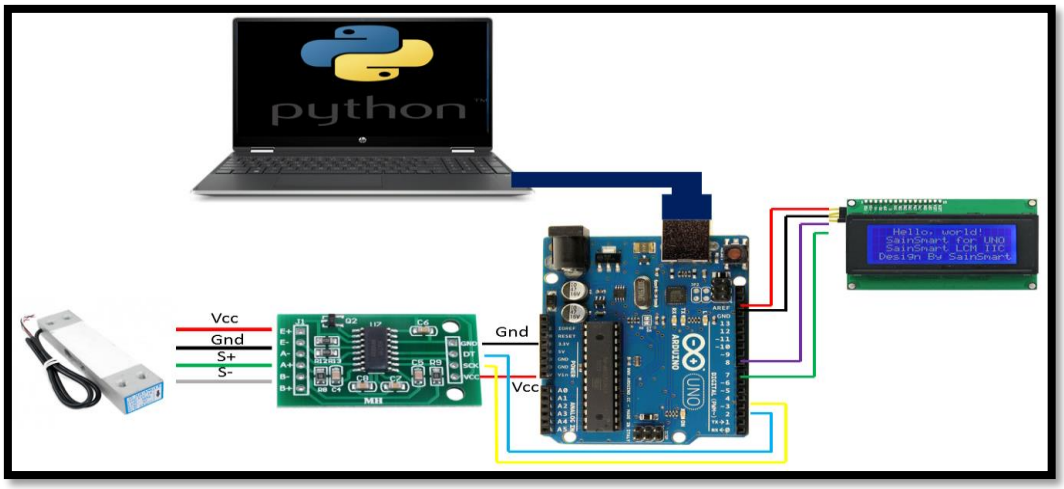


Figura 42. Diagrama pictográfico del sistema de extracción de datos de la balanza
Fuente: UNAC (2019); Autoría Propia.

Posteriormente se pasó a implementarlo en una placa de tipo galleta, se adiciono el núcleo de ferrita, sus 2 borneras y el termo contraíble en los cables de la celda de carga, para reducir el ruido eléctrico al cual está expuesto la salida de la celda de carga, adicionalmente se le adiciono un conector para energizar el sistema. (Véase la Figura N° 43, en la página 70)

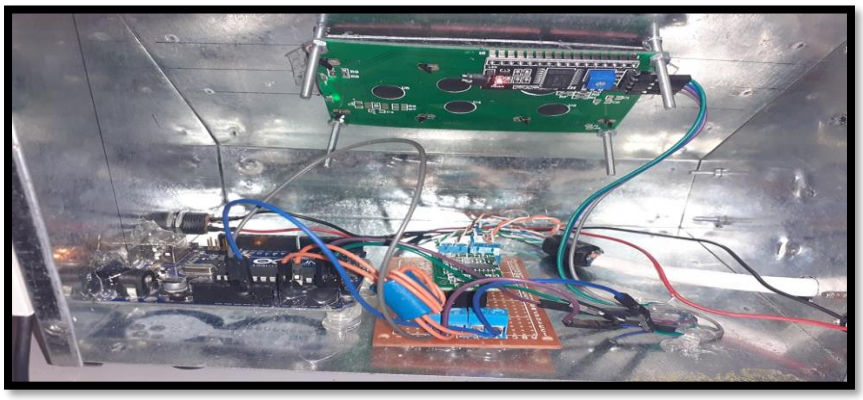


Figura 43. Conexiones físicas entre todos los componentes del sistema
Fuente: UNAC (2019); Autoría Propia.

Para proteger todo nuestro sistema electrónico de posibles golpes, ingreso de polvo y evitar que los cables de la celda de carga se muevan, se tuvo que diseñar una caja de metal en el software de AutoCAD, considerando las medidas oficiales de cada componente. (Véase la Figura N° 44)

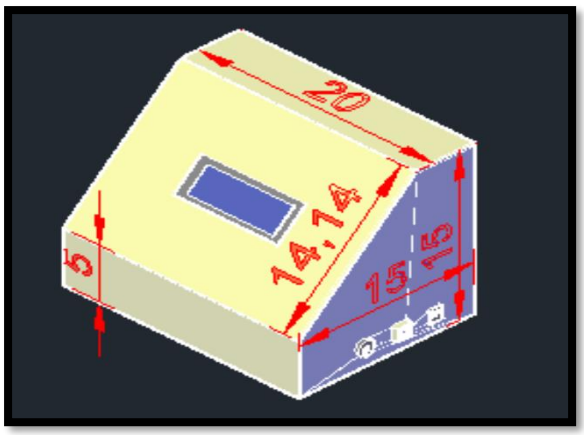


Figura 44. Diseño de caja de protección electrónica en AutoCAD
Fuente: UNAC (2019); Autoría Propia.

La caja cuenta con una entrada USB de tipo B, otra entrada por donde pasa el cable recubierto con termocontraíble que conecta la balanza con el módulo HX710B y una tercera entrada donde se conecta una tira de leds que más adelante iremos explicando para que es. y adicionalmente la caja cuenta con una ranura para colocar la pantalla LCD 20x4. (Véase la Figura N° 45)

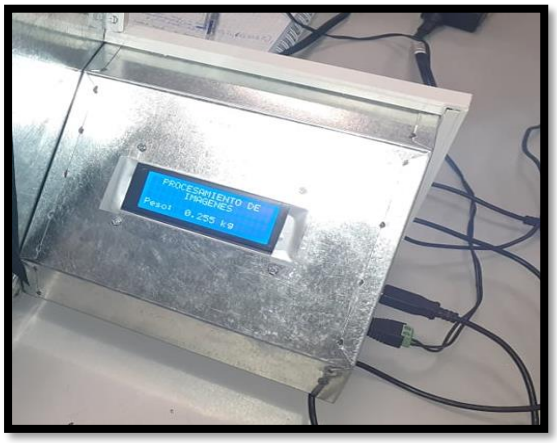


Figura 45. Caja de protección electrónica
Fuente: UNAC (2019); Autoría Propia.

- **Programación**

```
#include "HX711.h"
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 20, 4);
#define DOUT  A0
#define CLK  A1

HX711 balanza;

void setup()
{
  Serial.begin(9600);
  balanza.begin(DOUT, CLK);
  balanza.set_scale(340230);
  balanza.tare();
  lcd.clear();
  lcd.init();
  lcd.backlight();
}

void loop()
{

  lcd.setCursor(0, 0);
  lcd.print("  PROCESAMIENTO DE   ");
  lcd.setCursor(0, 1);
  lcd.print("      IMAGENES     ");
  lcd.setCursor(0, 3);
  lcd.print("Peso: ");
  lcd.setCursor(7, 3);
  lcd.print(balanza.get_units(),3);
  lcd.setCursor(12, 3);
  lcd.print(" kg");
  Serial.println(balanza.get_units(),3);
  if (Serial.available())
  {
    char temp = Serial.read();
    if (temp == 't' || temp == 'T')
    {
      balanza.tare();
      Serial.println("TARA");
    }
  }
```

}

El presente programa fue desarrollado en el software “ARDUINO IDE”, utilizando el hardware “ARDUINO UNO”, inicializamos importando las librerías de la celda de carga, la del LCD 20x4 y la del módulo I2C (HX711a.h, Wire.h y LiquidCrystal_I2C.h) respectivamente, cabe precisar que existe una compatibilidad entre el módulo HX710B y la librería HX711.h, se optó por utilizar el módulo HX710B por ser un chip más rápido que su versión posterior HX711A.

Posteriormente se procedió a indicar las dimensiones del LCD (20x4) mediante el comando LiquidCrsytal_I2C lcd (0x27, 20, 4), luego se declaró los pines DOUT A0 y CLK A1 que son los pines por donde sale la información del peso en bits y la señal de reloj respectivamente.

Bueno una vez ya declarado todos nuestros pines y configuradas nuestras librerías procedemos a trabajar en el void setup (), iniciamos una comunicación serial entre el Arduino y la PC mediante el comando Serial. Begin (9600), 9600 indica la velocidad de transmisión de datos entre los componentes descritos anteriormente, luego con el comando balanza. Begin (DOUT,CLK) le estamos inicializando la comunicación entre el Arduino y el Chip HX710B, con el comando balanza.set_scale(340230) es el factor de calibración de la celda de carga, hay que recordar que el ADC captura los valores en las unidades de bits y nosotros requerimos en gramos por lo cual hace necesario calcular el factor de calibración mediante la ecuación 36.

$$Fact_{cal} = \frac{Peso}{ADC} \quad \textbf{(36)}$$

El $Fact_{cal}$: Es el factor de calibración del sistema, sus unidades son (gr/bits), $Peso$: Es el peso de una pesa patrón certificada por el INACAL (Instituto NAcional de Calibración), ADC: Es el valor en bits que asigna el ADC del HX710B.

El comando balanza.tare(), sirve para Tarar (Considerar como 0gr cualquier objeto que se encuentre encima de la celda de carga al momento de encender la balanza), es muy útil pues como se explicó anteriormente el funcionamiento de una celda de carga se basa en el principio de deformación elástica por lo cual la celda de carga debe encontrarse suspendida dentro de la balanza, para lo cual se requiere que la celda este fijada a una estructura o una base, la cual evidentemente tiene un peso por es necesario considerarlo como TARA. Los comandos lcd.clear(),lcd.init() y lcd.backlight(), son comandos para inicializar la pantalla LCD de 20x4.

La librería “HX711.h” se encarga básicamente de 2 cosas, primero a través del pin CLK tiene que generar una señal de 1KHz la cual va directamente al módulo HX710B y segundo debe correr un ciclo for desde la posición 0 hasta la 23 para poder almacenar los 24bits que entrega el ADC a través del pin DOUT para luego aplicar el método de multiplicaciones sucesivas para recomponer el numero en base 10. Finalmente, el void loop () solo se encarga de mostrar el peso en el LCD y enviar dicho dato al Python a través de la comunicación serial.

- **Diseño de un prototipo de hardware a escala para la captura de imágenes**

Para el diseño del prototipo se consideró que la adquisición de las imágenes es una de las etapas fundamentales dentro del proceso de visión artificial, ya que, si no se realiza correctamente, el sistema no tienes las herramientas para procesar las imágenes de manera correcta.

El prototipo del sistema, así como el montaje general de un sistema de visión artificial se diseñó considerando la participación de cuatro componentes básicos: un sistema de iluminación, el hardware de captura (Cámara), una computadora y un software para el procesamiento de las imágenes capturadas (véase la Figura N° 46).

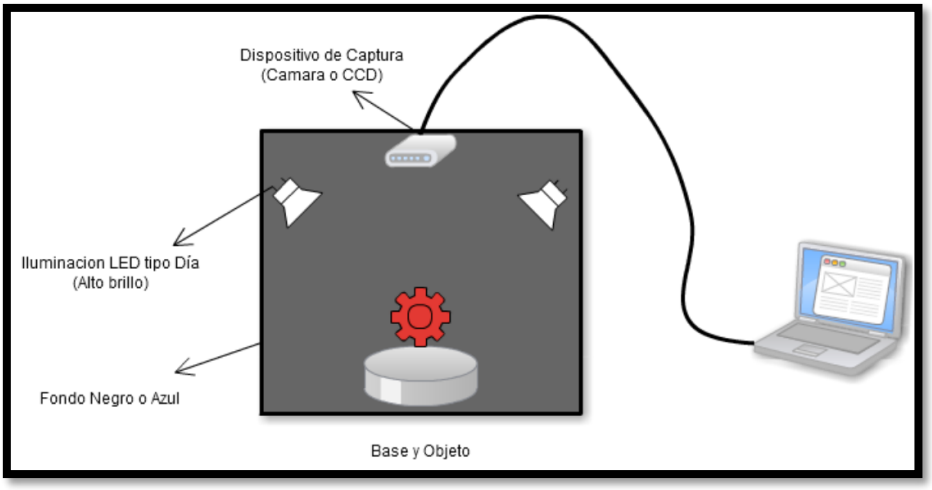


Figura 46. Componentes de un sistema de visión artificial.
Fuente: UNAC (2019); Autoría Propia.

Antes de mandar implementar el prototipo de hardware para la captura de imágenes, se diseñó este en AutoCAD. (Véase Figura N° 47)

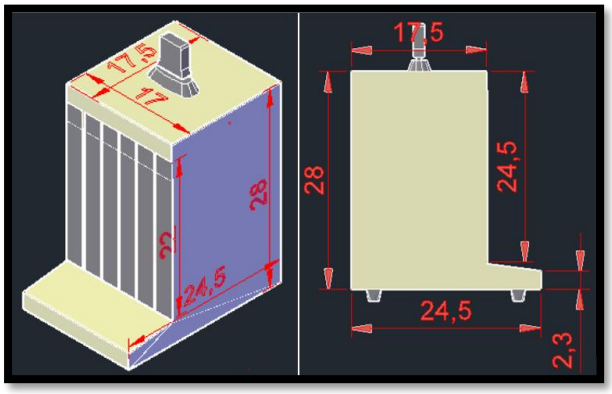


Figura 47. Diseño mecánico del prototipo de AutoCAD
Fuente: UNAC (2019); Autoría Propia.

Posteriormente se mandó a implementar el prototipo de hardware de captura de imágenes y se le adiciono el sistema de adquisición de datos de una balanza. (Véase la figura N° 48).

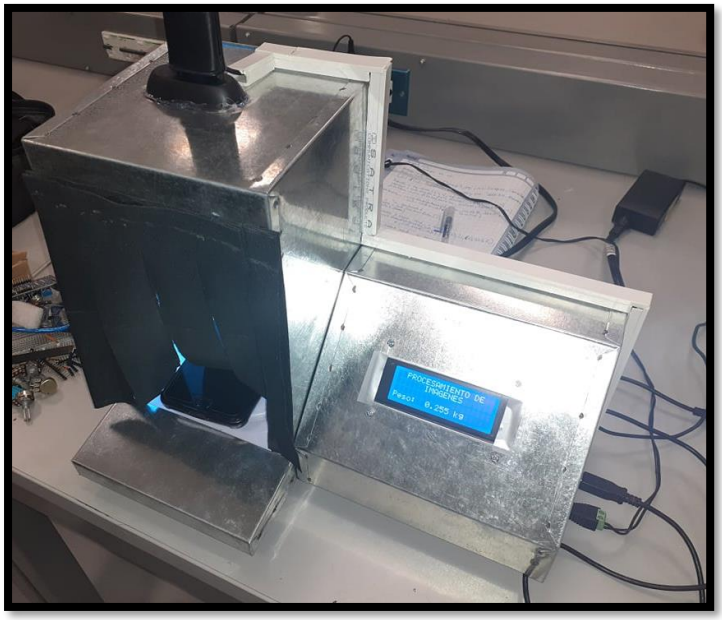


Figura 48. Implementación del prototipo de hardware de captura de imágenes.
Fuente: UNAC (2020); Autoría Propia.

El diagrama de secuencia del sistema (véase la Figura N° 49)

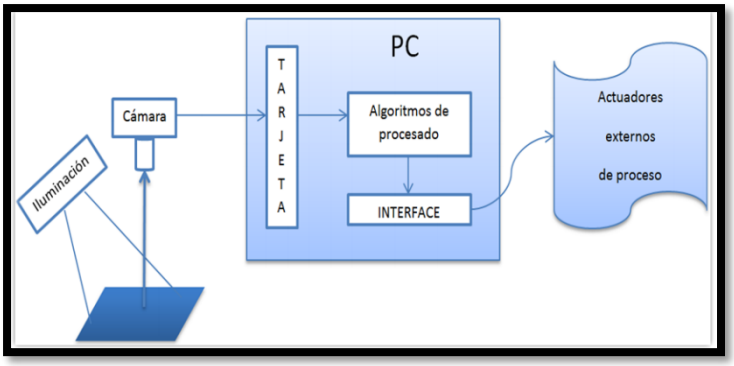


Figura 49. Diagrama sistema de visión artificial.
Fuente: UNAC (2020); Autoría Propia.

- **Desarrollo de un software de técnicas de procesamiento de imágenes para reconocer el color de los mangos**

Un algoritmo de visión artificial, cuenta con unos pasos que determinan la funcionalidad del sistema, para el desarrollo de este objetivo, se toma este algoritmo como modelo, que es la base para el diseño general del software. (Véase la Figura N° 50).

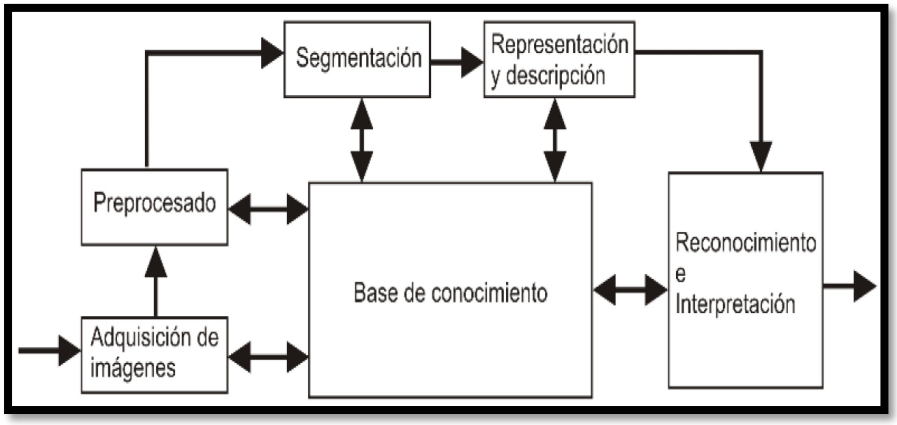


Figura 50. Esquema de sistema de visión artificial.
Fuente: UNAC (2019); Autoría Propia.

Antes de iniciar con la creación del software de reconocimiento de colores es necesario instalar las siguientes librerías: OPENCV (Procesamiento de imágenes), PILLOW y TKINTER (Crear Interfaces Graficas), Serial (Comunicación Serial), luego hacemos el llamado de las librerías en la cabecera del programa. (Véase la Figura N°51)

```
import tkinter as tk
from tkinter import ttk
from tkinter import*
from PIL import Image, ImageTk
import numpy as np
from tkinter import filedialog
from tkinter import messagebox
import serial
import sys,glob
import random
import threading
import cv2
import time
global camara
```

Figura 51. Llamado de las librerías OPENCV, PILLOW, TKINTER Y PySERIAL
Fuente: UNAC (2020); Autoría Propia.

La librería de OPENCV en Python se declara CV2, ***"from PIL import Image, ImageTK"*** es para poder colocar imágenes dentro del Tkinter, ***"import numpy as np"*** es para poder trabajar las imágenes desde un punto de vista matricial**,** ***"import serial.tools.list_ports"*** es para poder

testear todos los puertos serie disponibles en la computadora o laptop, ***"from tkinter import filedialog"*** es para poder guardar y cargar archivos de tipo blog de notas (.txt) en la interfaz y por último el ***"from tkinter import messagebox"*** es para poder mostrar mensajes en una ventana diferente.

Posteriormente creamos una variable global llamada cámara que nos permitirá ser utilizada en cualquier momento del programa, luego escribimos ***"cámara= cv2.VideoCapture(1)"*** que nos permite seleccionar la webcam con la cual deseamos trabajar como nosotros estamos trabajando con una laptop la webcam que viene incorporada se le asigna el valor 0 y como estamos utilizando la webcam del sistema extractor de datos de la balanza se le asigna el valor 1, Ahora creamos una ventana en Tkinter mediante el comando ***"ventana = tk.Tk()"***, tenemos que ponerle un título a la ventana mediante el comando ***"ventana.title("RECONOCIMIENTO DE COLOR DE MANGOS")***", luego tenemos que seleccionar el color de fondo de la interfaz mediante el comando ***"ventana.config(bg = "black")"***, bg significa background que es color de fondo, nosotros seleccionamos el color negro, luego tenemos que definir el tamaño por defecto que tendrá la interfaz gráfica con el comando ***"ventana.geometry('1300x580')"***, nuestra interfaz por defecto tiene un tamaño de 1300 x580 pixeles (Véase la Figura N° 52)

```
global camara
camara= cv2.VideoCapture(0)

ventana = tk.Tk()
ventana .title("RECONOCIMIENTO DE COLOR DE MANGOS")
ventana .config(bg = "black")
ventana .geometry('1300x580')
miframe=Frame(ventana).pack()

texto =Label(miframe, text="AUTOMATIZACIÓN DEL PROCESO ",bg ="black",fg = "white",
                  font =("Comic Sans MS",13)).place(x=85,y=30)

texto1 =Label(miframe, text="DE SELECCIÓN DE MANGOS",bg ="black",fg = "white",
                  font =("Comic Sans MS",13)).place(x=100,y=60)

imagen = PhotoImage(file="unac.png")
Label(ventana, image = imagen, bd=0).place(x=10,y=0)
```

Figura 52. Programación para la creación de la venta principal
Fuente: UNAC (2020); Autoría Propia.

Los comandos de texto y texto1 son para poder colocarle un título a la interfaz que muy diferente que colocarle un título a la venta como se explicó anteriormente, la ventaja que te da el Tkinter es que puedes manejar las propiedades de dicho texto como el color de la letra, la fuente, el tamaño y la posición, el comando ***"imagen=PhotoImage(file="unac.png")"*** permite cargar una imagen a nuestra interfaz gráfica, el único requisito es que la imagen este guardada en la misma carpeta del archivo principal (Véase la Figura N° 53)

Figura 53. Interfaz gráfica TKINTER en PYTHON
Fuente: UNAC (2020); Autoría Propia.

Ahora creamos un algoritmo para poder enlazarnos al Arduino mediante el puerto serie y la creación de un ***"ComboBox"*** y así poder extraer el valor de la balanza, para lo cual empezaremos corriendo un ciclo ***"for"*** que inicie en 1 hasta 256, el cual se ira cargando uno a uno en la variable ***"s"*** mediante el siguiente comando ***"s = serial.Serial('COM'+str(i),9600)"***, de esta manera se va generando los puertos, luego se espera aproximadamente 2 segundos y se revisa que puertos están disponibles en el Arduino mediante el siguiente comando ***"available.append((s.portstr))"*** y si en caso no hubiera ningún puerto disponible se salta a la siguiente línea. (Véase la Figura N° 54)

```
for i in range(256):
    try:
        s = serial.Serial('COM'+str(i),9600)
        time.sleep(2)
        available.append( (s.portstr))

    except serial.SerialException:
        pass
```

Figura 54. Programación para buscar puertos
Fuente: UNAC (2020); Autoría Propia.

Ahora creamos un ***"ComboBox"*** en nuestra interfaz gráfica para poder visualizar los puertos y se pueda ejecutar el programa explicado en la Figura N° 55. Para la creación del combobox utilizamos el siguiente comando ***"listas_arduino = ttk. Combobox(ventana,width=30, state="readonly")"***, luego creamos una lista donde podemos visualizar los puertos disponibles y conectarnos a los mismos. Iniciamos con un for que va desde ***"s"*** hasta todos los puertos disponibles y eso lo muestra en el ***"comboBox"***, si detecta que el puerto no está vacío, espera 500 milisegundos y salta a la función ***"sensor_tem"***. (Véase la Figura N° 56)

```
listas_arduino = ttk.Combobox(ventana,width=30, state="readonly")
listas_arduino.place(x=210, y=141)


def listaArduino():
    for s in available:
        #print ("%s" % (s))
        listas_arduino['values'] =(s)
        listas_arduino.set(s)
        if s!=" ":
            sensor.after(500, sensor_tem)
```

Figura 55. Programación para crear un COMBOBOX y enlazar el puerto serie
Fuente: UNAC (2020); Autoría Propia.

Ahora nos situamos en la función ***"sensor_tem"***, creamos la variable dato y mediante el comando ***"dato = s.readline()"*** capturamos el dato transmitido por el puerto serie y con el siguiente comando ***"senso = dato[0:5]"*** definimos el ancho del dato un entero, una coma y 3 decimales, con el comando ***"readingt.set(senso)"*** lo repetimos nuevamente con pausa de 50 milisegundos y con el comando ***"eadingt = StringVar()"*** convertimos el peso que actualmente se encuentra en formato ***"char"*** a un formato decimal. (Véase la Figura N° 56)

```
def sensor_tem():
    dato = s.readline()
    senso = dato[0:5]
    readingt.set(senso)
    sensor.after(50, sensor_tem)

readingt = StringVar()
```

Figura 56. Programación para capturar valores del puerto serie
Fuente: UNAC (2020); Autoría Propia.

Se ejecuta el programa y se observa el ***"comboBox"*** para seleccionar un puerto y con un botón de ***"Conectar Puerto"*** nos enlazamos con el Arduino y podemos visualizar el peso de la balanza. (Véase la Figura N° 57)

Figura 57. Interfaz gráfica del seleccionador de puertos
Fuente: UNAC (2020); Autoría Propia.

Ahora creamos un botón para conectar la cámara, para esto creamos la variable conectar con el comando **"*Button*",** la ventaja que nos da este comando es que podemos configurar propiedades como: Texto, ancho, color, tipo de letra, tamaño, posición y el más importante ***"command"*** que es la función a la cual va saltar el programa cuando el botón conectar es presionado que en nuestro caso es ***"camara1"*** (Véase la Figura N° 58)

```
def camara1():
    conectar2()
    ventana_camara.pack(anchor=tk.SE)

conectar = Button(miframe, text ="CONECTAR", relief="raised", borderwidth=5,
                  command = camara1,bg ="green3",fg = "black",
                  font =("Comic Sans MS",12)).place(x=210,y=60)
```

Figura 58. Programación para enlazar la webcam a PYTHON
Fuente: UNAC (2020); Autoría Propia.

De la función ***"camara1"*** salta a la función ***"conectar2"***, luego el programa analiza si la webcam está disponible mediante el un sentencia de control ***"if"*** y si está disponible realiza una lectura mediante el comando ***"_,frame=camara.read()"*** y nos entrega como salida 2 parámetros: uno que es audio y no lo utilizamos en nuestra aplicación por eso lo declaramos como **"_"**, en Python si una variable no va a ser utilizada esta es declarada como guion abajo y la segunda es ***"frame"*** que es el video que en realidad son imágenes solo que las pasan tan rápido que da la impresión de que es video, luego utilizamos el comando ***"cv2. cvtColor"*** para poder realizar la conversión de espacio vectorial RGB a HSV para poder realizar el seguimiento del objeto por color. (Véase la Figura N° 59)

```
def conectar2():
    global camara
    if camara.isOpened() == True:
        _,frame = camara.read()
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
```

Figura 59. Función para enlazar la webcam a PYTHON
Fuente: UNAC (2020); Autoría Propia.

Ahora creamos un botón para desconectar la webcam, para esto creamos la variable conectar con el comando **"*Button*"** cuyas ventajas ya las

explicamos anteriormente, al presionar el botón se salta a la función ***"cámara_des"*** y utilizamos el comando ***"camara.release()"*** para desconectar la webcam y el comando ***"ventana_camara.place(x=645,y=-600)"*** indica la posición donde se encuentra la webcam que se va a desconectar. (Véase la Figura N° 60)

```
def camara_des():
    camara.release()
    ventana_camara.place(x=645,y=-600)

desconectar = Button(miframe, text ="DESCONECTAR", relief="raised", borderwidth=5 ,
                     command = camara_des ,bg ="firebrick1",fg = "black",
                     font =("Comic Sans MS",12)).place(x=355,y=60)
```

Figura 60. Botón para desconectar la webcam
Fuente: UNAC (2020); Autoría Propia.

Al ejecutar el programa podemos notar la creación que efectivamente se han creado 2 botones, uno de color rojo para enlazar la aplicación con webcam y uno de color verde para romper dicho enlace. (Véase la Figura N° 61)

Figura 61. Interfaz gráfica para conectar y desconectar la webcam
Fuente: UNAC (2020); Autoría Propia.

Ahora creamos una ventana donde colocaremos los ***"SLIDER's"*** que varían de 0-255, los cuales nos permitirán calibrar el programa a los colores deseados que para nuestro caso son el amarrillo y el verde mediante la

transformación de espacio vectorial RGB a HSV, por eso surge la necesidad de separar los slider en 2 grupos (6 para amarillo y 6 para el verde). (Véase la Figura N° 62)

```
notebook = ttk.Notebook(ventana)
notebook.place(x=20 ,y=230)
pes1 = ttk.Frame(notebook,height=310,width=265)
pes2 = ttk.Frame(notebook,height=310,width=265)
notebook.add(pes1, text='COLOR 1')
notebook.add(pes2, text='COLOR 2')
```

Figura 62. Creación de un espacio para 2 grupos de SLIDER'S
Fuente: UNAC (2020); Autoría Propia.

Ahora pasamos a crear los ***"SLIDER'S"***, para lo cual utilizamos el comando ***"w1_1=tk.Scale(pes1,label='TonoMin',from_=0,to=255,length=260,width=9,orient=HORIZONTAL)"***, que nos permite crear un slider que varía de 0-255, dicho valor se guarda en la variable ***"w1_1"*** que está situado en ***"pes1"*** que pertenece al grupo de color1 y lo mismo se repite 6 veces para el grupo pes1 y pes2 respectivamente. (Véase la Figura N° 63)

```
w1_1 = tk.Scale(pes1, label='Tono Min',from_=0, to=255,
                length=260,width=9,orient=HORIZONTAL)
w1_1.place(x=0,y=0)

w2_1 = tk.Scale(pes1, label='Tono Max',from_=0, to=255,
                length=260,width=9,orient=HORIZONTAL)
w2_1.place(x=0,y=50)

w3_1= tk.Scale(pes1, label='Brillo Min',from_=0, to=255,
                length=260,width=9,orient=HORIZONTAL)
w3_1.place(x=00,y=100)

w4_1 = tk.Scale(pes1, label='Brillo Max',from_=0, to=255,
                length=260,width=9,orient=HORIZONTAL)
w4_1.place(x=0,y=150)

w5_1 = tk.Scale(pes1, label='Saturación Min',from_=0, to=255,
                length=260,width=9,orient=HORIZONTAL)
w5_1.place(x=0,y=200)

w6_1 = tk.Scale(pes1, label='Saturación Max',from_=0, to=255,
               length=260,width=9,orient=HORIZONTAL)
w6_1.place(x=0,y=250)
```

Figura 63. Creación de 6 SLIDER'S
Fuente: UNAC (2020); Autoría Propia.

Ahora creamos un botón para guardar los valores de los ***"SLIDER'S"*** en un archivo block de notas (.txt) y cuando el botón sea presionado salte a la función llamada ***"guardarHSV"***. (Véase la Figura N° 64)

```
guardar_txt = Button(miframe, text ="GUARDAR", relief="raised", borderwidth=5,
                     bg ="gray35",fg = "white", command = guardarHSV,
                     font =("Comic Sans MS",12)).place(x=370,y=320)
```

Figura 64. Programación para la creación de un botón de guardo
Fuente: UNAC (2020); Autoría Propia.

Cuando el programa salta a la función ***"guardarHSV"***, creamos una ventana de Windows donde nos permitirá seleccionar la ruta y el nombre del archivo en formato ***".txt"***, luego capturamos los valores de los slider's en las variables ***(tmin, tmax, bmin, bmax, smin, smax)*** correspondiente a los 6 primeros slider's y ***(tomin, tomax, brmin, brmax, samin, samax)*** correspondiente a los otros 6 slider's. Luego creamos una variable de tipo string para poder formar una cadena de valores (dato, salto de línea, dato2, salto de línea, etc.), ese procedimiento se realiza unas 12 veces, 6 por cada grupo de slider's. (Véase la Figura N° 65)

```
def guardarHSV():
    archivo = filedialog.asksaveasfilename(initialdir = "/", title = "Guardar como",filetypes =
    (("Documentos de texto","*.txt"),("todos los archivos","*.*")), defaultextension=".txt")
    if archivo!='':
        archivo = open(archivo, 'w')

        tmin=w1_1.get()
        tmax=w2_1.get()
        bmin=w3_1.get()
        bmax=w4_1.get()
        smin=w5_1.get()
        smax=w6_1.get()

        tomin=w1_2.get()
        tomax=w2_2.get()
        brmin=w3_2.get()
        brmax=w4_2.get()
        samin=w5_2.get()
        samax=w6_2.get()
        variable_string ="{}" "\n" "{}" "\n" "{}" "\n" "{}" "\n" "{}" "\n" "{}" "\n"  "{}" "\n"
        archivo.write(variable_string)
        archivo.close()
        messagebox.showinfo("ATENTO !! ","Tus valores del HSV fueron guardados")
```

Figura 65. Programación de la función guardarHSV
Fuente: UNAC (2020); Autoría Propia.

Ejecutamos el programa, seleccionamos nuestros valores HSV y hacemos click en el botón de guardar y nos aparece una ventana donde seleccionamos nuestra ruta de guardado. (Véase la Figura N° 66)

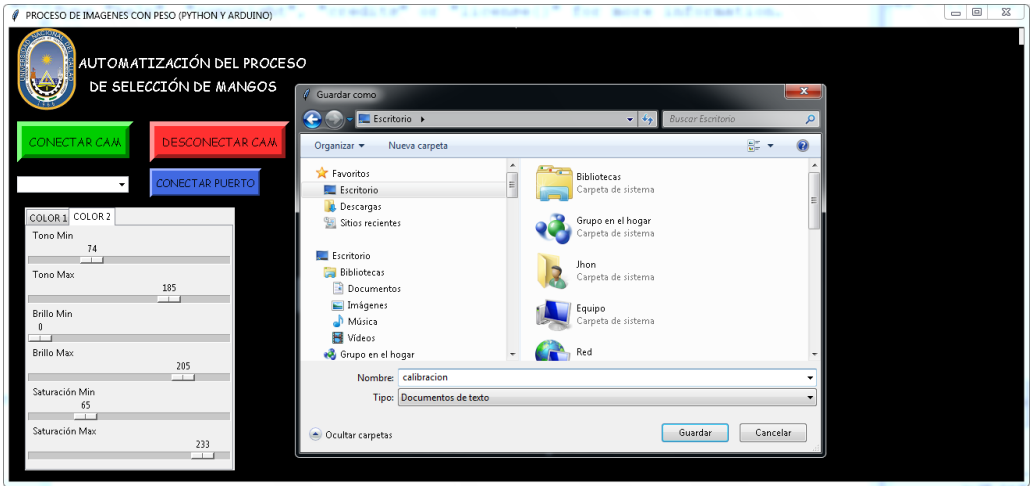


Figura 66. Prueba del botón guardado
Fuente: UNAC (2020); Autoría Propia.

Ahora creamos un botón cuyo nombre es ***"ABRIR",*** para poder cargar la calibración de los parámetros HSV en los slider's, al presionar dicho botón el programa salta a la función ***"mostrar".*** (Véase la Figura N° 67)

```
ajustar_hsv = Button(miframe, text ="ABRIR", relief="raised", borderwidth=5,
                     bg ="gray20",fg = "white", command = mostrar,
                     font =("Comic Sans MS",12)).place(x=500,y=320)
```

Figura 67. Programación para crear el botón abrir.
Fuente: UNAC (2020); Autoría Propia.

Cuando el programa salta a la función ***"mostrar"*** se abre una ventana de Windows que nos pide ubicar el archivo que se desea cargar en los slider's mediante el uso del comando ***"SET"*** y las variables ***(w1_1, w2_1, w3_1, w4_1, w5_1, w6_1)*** para los primeros 6 slider's y ***(w1_2, w2_2, w3_2, w4_2, w5_2, w6_2)*** (Véase la Figura N° 68)

```
def mostrar():
    archivo = filedialog.askopenfilename(initialdir = "/", title = "Seleccione archivo",
    filetypes = (("Documentos de texto","*.txt"),("todos los archivos","*.*")),defaultextension=".txt")
    if archivo!='':
        archivo_ = open(archivo, 'r')
        contenido = []

        for linea in archivo_.readlines():
            contenido.append(linea.split())

        w1_1.set(contenido[0][0])
        w2_1.set(contenido[1][0])
        w3_1.set(contenido[2][0])
        w4_1.set(contenido[3][0])
        w5_1.set(contenido[4][0])
        w6_1.set(contenido[5][0])
        w1_2.set(contenido[6][0])
        w2_2.set(contenido[7][0])
        w3_2.set(contenido[8][0])
        w4_2.set(contenido[9][0])
        w5_2.set(contenido[10][0])
        w6_2.set(contenido[11][0])
```

Figura 68. Programación de la función mostrar
Fuente: UNAC (2020); Autoría Propia.

Ejecutamos el programa y podemos notar que al hacer click en el botón ***"ABRIR"*** podemos notar que se abre una ventana que nos pide ubicar nuestro archivo (.txt), una vez ubicado el archivo los valores se cargan en nuestros slider's (Véase la Figura N° 69)

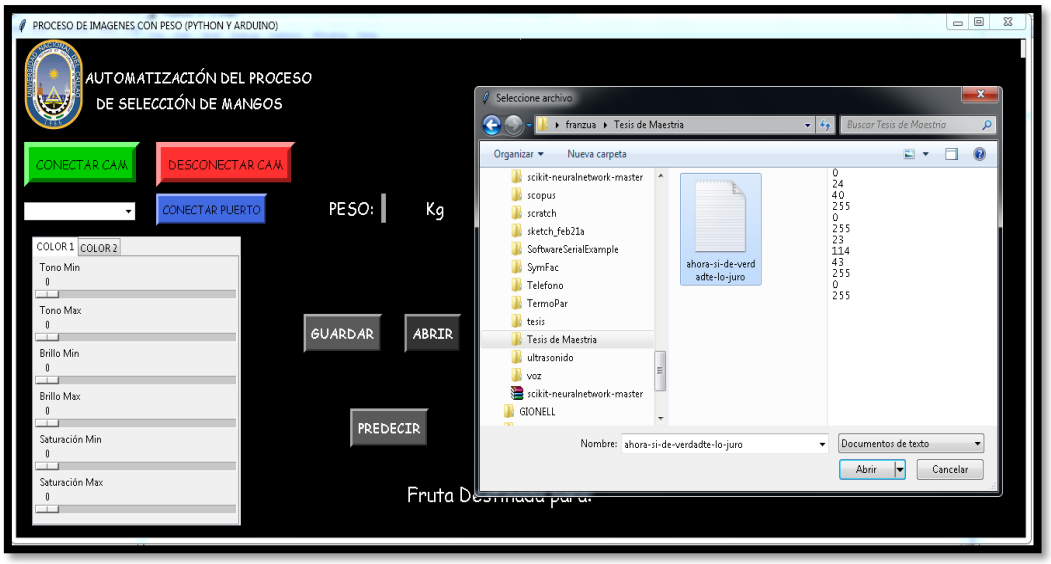


Figura 69. Prueba del botón abrir.
Fuente: UNAC (2020); Autoría Propia.

El programa principal inicia definiendo una variable de tipo global llamada cámara, seguido de un condicional de tipo if ***"if camara.isOpened() == True:"*** si la afirmación de que la cámara ha sido encendida es verdadera se procede a realizar una lectura de la cámara mediante el siguiente comando ***"_,frame = camara.read()"*** el cual nos devuelve 2 parámetros uno es audio que no lo vamos a utilizar por eso lo representamos con ***"_"*** y ***"frame"*** que es el video de la cámara, luego convertiremos la variable ***"frame"*** que se encuentra en formato RGB al formato HSV para poder realizar de una manera más sencilla el proceso de selección por color. Ahora nuestro video ya no se llama ***"frame"*** sino ***"HSV".*** (Véase la Figura N° 70)

```
def conectar2():
    global camara
    if camara.isOpened() == True:
        _,frame = camara.read()
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
```

Figura 70. Programación para encender la webcam y convertir imágenes de RGB A HSV
Fuente: UNAC (2020); Autoría Propia.

Para realizar el proceso de selección por color aplicando la técnica de transformación de espacio vectorial RGB a HSV es necesario establecer los valores HSV, para eso utilizaremos el comando ***"get()"*** para poder capturar dichos valores de los ***"SLIDER'S"*** y guardándolos en las variables: ***tmin, tmax, bmin, bmax, smin y smax,*** luego los agrupamos en 2 matrices llamadas ***"minimo_1"*** y ***"máximo_1"***, en la primera matriz guardaremos los valores: ***tmin, bmin y smin,*** y en la segunda: ***tmax, bmax y Smax,*** y de esta manera queda definido el espacio vectorial HSV, ahora hay que cargar estas dos matrices en nuestra video llamado "***HSV"*** mediante el comando ***"mascara = cv2.inRange(hsv,minimo_1,maximo_1)"***, ahora nuestro video se llama ***"mascara"***, ahora pasamos a eliminar todo el ruido que se presenta en nuestras imágenes, para eso vamos a definir una matriz llamada ***"kernel1_1"*** conformada por números 1 de 3x3 mediante el siguiente

comando ***"kernel_1= np.ones((3,3),np.uint8)",*** esta matriz lo que hace es posicionar su elemento central por toda la imagen pixel por pixel y si al menos un elemento de su alrededor es un 1 este pixel central se convierte en cero, esa la definición del proceso morfológico de erosión el cual se utiliza para eliminar el ruido mediante el siguiente comando ***"erosion_1= cv2.erode(mascara,kernel_1,iterations =3)"***, la ventaja que nos da python con respecto a otros programas es que nos permite indicar el número de veces que queremos que se aplique este filtro, la desventaja es que al aplicarlo la imagen se deforma en los contornos, por eso es recomendable que luego de aplicar el filtro de erosión debe seguir el de dilatación que consiste en que si uno de los elementos de alrededor del ***"kernel_1"*** es un 1 pues el elemento central también se convierte en 1, de esta manera el ruido que ya fue eliminado no puede volver a aparecer y la figura principal recupera su forma original de cierta forma mediante el comando ***"dilation_1 = cv2.dilate(erosion_1,kernel_1,iterations = 2)"***, este procedimiento lo aplicamos 2 veces porque intentemos seleccionar 2 tipos de colores. (Véase la Figura N° 71)

```
# .....................PRIMER COLOR.........#
tmin=w1_1.get()
tmax=w2_1.get()
bmin=w3_1.get()
bmax=w4_1.get()
smin=w5_1.get()
smax=w6_1.get()
minimo_1= np.array([tmin,smin,bmin])
maximo_1= np.array([tmax,smax,bmax])
mascara = cv2.inRange(hsv,minimo_1,maximo_1)
kernel_1= np.ones((3,3),np.uint8)
erosion_1= cv2.erode(mascara,kernel_1,iterations = 3)
dilation_1 = cv2.dilate(erosion_1,kernel_1,iterations = 2)

#............SEGUNDO COLOR.............#
tomin=w1_2.get()
tomax=w2_2.get()
brmin=w3_2.get()
brmax=w4_2.get()
samin=w5_2.get()
samax=w6_2.get()
minimo_2= np.array([tomin,samin,brmin])
maximo_2= np.array([tomax,samax,brmax])
mascara_2 = cv2.inRange(hsv,minimo_2,maximo_2)
kernel_2= np.ones((3,3),np.uint8)
erosion_2= cv2.erode(mascara_2,kernel_2,iterations = 3)
dilation_2 = cv2.dilate(erosion_2,kernel_2,iterations = 2)
```

Figura 71. Programa para transformar video RGB a HSV y eliminación de ruido
Fuente: UNAC (2020); Autoría Propia.

Ahora vamos a dibujar los contornos para poder tener un indicador visual que nos valide nuestros parámetros HSV, luego calcularemos sus áreas y la figura que tenga mayor área determinara la predominancia de la fruta (verde y amarillo). Primero vamos a buscar los contornos de la variable ***"dilation_1"*** para eso utilizaremos el comando ***"findContours"*** y la línea de programación ***"contornos, _ = cv2.findContours (dilation_1,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)"*** donde ***"dilation_1"*** es la imagen a la cual le vamos a buscar sus contornos, ***"cv2.RETR_EXTERNAL"*** es el modo de recuperación de contornos, se recupera los contornos exteriores extremos y ***"cv2.CHAIN_APPROX_SIMPLE"*** es el método de aproximación de contornos, comprime segmentos horizontales, verticales y diagonales y deja solo sus puntos finales. Por ejemplo, un contorno rectangular superior derecho está codificado con 4 puntos, este comando se aplica 2 veces uno para ***"dilation_1"*** como ya hemos explicado y otra vez para ***"dilation_2"***. (Véase la Figura N° 72)

```
contornos,_ = cv2.findContours(dilation_1,cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
contorn,__ = cv2.findContours(dilation_2,cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

Figura 72. Programación para buscar contornos
Fuente: UNAC (2020); Autoría Propia.

Ahora procederemos a buscar áreas y a dibujarlas, para lo cual usaremos con un ciclo "***for"*** que inicializa en C =0 y va hasta la variable ***"contornos",*** esto nos permitirá ubicar todas las figuras que tengan contornos, luego utilizaremos el comando ***"area = cv2.contourArea(c)"*** para calcular el área de todos los elementos que tengan contornos, ahora solo trabajaremos con los elementos que se tengan una área mayor a 13000 pixeles y menor a 20 000 pixeles, los elemento que se encuentren fuera de este rango son catalogados como ruido y por no nos interesa trabajar con ellos ***"if area > 13000 and area < 200000:"***, ahora pasamos a dibujar el contornos de esas áreas usamos el comando ***"drawContours"*** en la

siguiente línea de programación ***"cv2.drawContours(frame,contornos,0,(255,255,0),2, cv2.LINE_AA)"***, donde ***"frame"*** es donde quieres que se dibuje los contornos, en este caso es el video que captura la webcam, ***"contornos"*** es la variable que definimos anteriormente para encontrar todos los contornos en la imagen, ***"0"*** es el contourIdx, parámetro que indica un contorno para dibujar. Si es negativo, se dibujan todos los contornos, ***"(255,255,0)"*** la primera posición pertenece al rojo, la segunda al verde y el tercero al azul, solo pueden variar de (0-255), por lo tanto, el color que se representa ahí es el amarillo, ***"2"*** indica el grosor de la línea que se va a dibujar y ***"cv2.LINE_AA"*** es la conectividad de la línea, este procedimiento se realiza 2 veces. (Véase la Figura N° 73).

```
for c in contornos:
    area = cv2.contourArea(c)
    fruta_madura = area
    if area > 13000 and area < 200000:
        cv2.drawContours(frame,contornos,0,(255,255,0),2, cv2.LINE_AA)
    continue
for a in contorn:
    area_1 = cv2.contourArea(a)
    fruta_verde = area_1
    if area_1 > 13000 and area_1 < 200000:
        cv2.drawContours(frame,contorn,0,(0,255,0),2, cv2.LINE_AA)
    continue
```

Figura 73. Programación para encontrar contornos y graficar áreas.
Fuente: UNAC (2020); Autoría Propia.

Ahora que ya tenemos calculado las áreas de la fruta (amarrillo y verde), vamos a colocar 2 condicionales, si el área de ***"fruta_madura"*** es mayor que ***"fruta_verde"*** enviamos el número "1" y si se da el caso contrario enviamos el número "0", para luego ser utilizado en nuestra neurona perceptrón. (Véase la Figura N° 74)

```
if fruta_madura > fruta_verde:
    color.set(1)
if fruta_madura < fruta_verde:
    color.set(0)
```

Figura 74. Programación para determinar la predominancia de una fruta.
Fuente: UNAC (2020); Autoría Propia.

Colocamos una fruta dentro de la balanza, corremos el sistema, calibramos nuestros valores HSV y le damos guardar a los mismos (Véase la Figura N° 75)

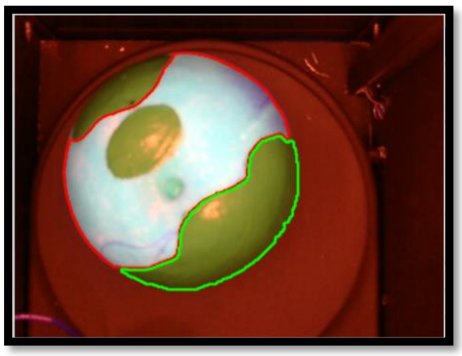


Figura 75. Prueba de selección de colores
Fuente: UNAC (2020); Autoría Propia.

Ahora creamos un botón cuyo nombre es ***"PREDECIR",*** que nos permite saltar a la función ***"neurona_perceptron", donde*** se ejecuta el algoritmo de una neurona perceptrón previamente entrenada para que pueda tomar una decisión sobre los datos de entrada: peso y color. (Véase la Figura N° 76)

```
predecir = Button(text ="PREDECIR", relief="raised", borderwidth=5,
                  bg ="gray35",fg = "white", command = neurona_perceptron,
                  font =("Comic Sans MS",12)).place(x=430,y=430)
```

Figura 76. Programación crear el botón predecir
Fuente: UNAC (2020); Autoría Propia.

Cuando el programa salta a la función ***"neurona_perceptron"*** lo primero que hacemos es definir una clase llamada ***"Perceptron"*** que a su vez va estar dividida principalmente en 3 partes: ***"__init__"*** que es el inicio donde

primero definimos el número de entradas que vamos a utilizar mediante la variable ***"input_number"*** y con el comando ***"self._ins = input_number"*** lo asociamos con la variable ***"self._ins"*** porque así nos lo pide la función ***"__init__"***, luego definimos los pesos sinápticos mediante la variable ***"w"*** y el comando ***"self._w = [random.random() for _ in range(input_number)]"*** de esta manera se logra que los pesos sinápticos sean aleatorios y lo asociamos a la variable ***"self._w"*** y por ultimo definimos el factor de aprendizaje mediante la variable ***"step_size"*** que para nuestro caso va ser 0.1 y que con el comando ***"self._eta = step_size"*** lo asociamos con la variable ***"self._eta"***.(Véase la Figura N° 77)

```
def neurona_perceptron():
    class Perceptron:
        def __init__(self,input_number,step_size = 0.1):
            self._ins = input_number
            self._w = [random.random() for _ in  range(input_number)]
            self._eta = step_size
```

Figura 77. Programación de creación de neurona perceptrón parte I
Fuente: UNAC (2020); Autoría Propia.

Parte 2, ahora vamos a definir la función ***"predict"*** cuyos elemento de entrada son (***"self"*** y ***"inputs"***) y lo que hacemos aquí es realizar una multiplicación entre las entradas y los pesos sinápticos y su posterior suma entre ellos mediante el comando ***"weighted_average = sum(w*elm for w,elm in zip (self._w,inputs))"*** y lo asociamos a la variable ***"weighted_average"***, luego ponemos una condición mediante el siguiente comando ***"if weighted_average > 0:"*** , en caso la afirmación sea verdadera nos retorna el número 1 y si en caso fuera falsa nos entrega el número 0. (Véase la Figura N° 78)

```
def predict(self,inputs):
    weighted_average = sum(w*elm for w,elm in zip (self._w,inputs))
    if weighted_average > 0:
        return 1
    return 0
```

Figura 78. Programación de creación de neurona perceptrón parte II
Fuente: UNAC (2020); Autoría Propia.

Parte 3, ahora vamos a definir la función ***"train"*** para realizar la etapa de entrenamiento de la neurona, que tiene como argumento ***("SELF","INPUTS"*** y ***"EX_OUTPUT")***, primero vamos a utilizar la función ***"predict"*** mediante el siguiente comando ***"output = self.predict(inputs)"***, para lograr obtener la salida, luego calculamos el ***"error"*** mediante el siguiente comando ***"error = ex_output - output"***, que es una resta entre ***"ex_output"*** que es la salida esperada y ***"output"*** que es la salida que obtuvimos de la función ***"predict"***, si en caso el error calculado es diferente de 0 quiere decir que a la neurona todavía le falta entrenar por lo tanto vamos a colocar un condicional mediante el siguiente comando ***"if error !=0:"*** , al notar la existencia de un error es necesario actualizar los pesos sinápticos mediante el siguiente comando ***"self._w = [w + self._eta*error*x for w,x in zip (self._w,inputs)]"***, que es la implementación de la ecuación **(2.17)** y si en caso no hubiera error significa que los pesos sinápticos calculados anteriormente son los correctos. (Véase la Figura N° 79)

```
def train (self,inputs,ex_output):
    output = self.predict(inputs)
    error = ex_output - output
    if error !=0:
        self._w = [w + self._eta*error*x for w,x in
        zip (self._w,inputs)]
    return error
```

Figura 79. Programación de creación de neurona perceptrón parte III
Fuente: UNAC (2020); Autoría Propia.

Una vez ya creada nuestra neurona es momento de cargarlo con la matriz de aprendizaje, para esto pedimos ayuda a un grupo de trabajadores que se encargan de clasificar los mangos, ellos nos proporcionaron 5 mangos de consumo local y 5 mangos para exportar, con los cuales armamos una matriz de 3x10 done la primera columna es peso, la segunda es color (1 amarillo y 0 verde) y la tercera columna es la salida (0 si es consumo local y 1 si es para exportar). (Véase la Figura N° 80)

```
input_data = [[0.263,0,0],
              [0.270,0,0],
              [0.252,0,0],
              [0.163,0,0],
              [0.231,0,0],
              [0.351,1,1],
              [0.266,1,1],
              [0.353,1,1],
              [0.333,1,1],
              [0.322,1,1]]
```

Figura 80. Matriz de aprendizaje de la neurona perceptrón.
Fuente: UNAC (2020); Autoría Propia.

Ahora tenemos que definir nuestro ***"Perceptron"*** de 3 entradas: la primera entrada es +1, es por el modelo de la neurona perceptrón, la segunda es el peso y el 3ero es el color, mediante el comando ***"pr = Perceptron (3)"***, ahora corremos un ciclo ***"for"*** para poder realizar el entrenamiento de la neurona, nosotros lo realizamos 15000 veces. (Véase la Figura N° 81)

```
pr = Perceptron(3)

for _ in range (15000):
    for  person in input_data:
        output = person[-1]
        inp = [1] + person[0:-1]
        err = pr.train(inp,output)
```

Figura 81. Programación para entrenar neurona.
Fuente: UNAC (2020); Autoría Propia.

Ahora para la parte final extraemos el dato del ***"peso"*** mediante el comando ***"peso = flota(entrada_peso.get())"***, y el dato del ***"color"*** mediante el siguiente comando ***"color = float(entr_fruta.get())"*** y por último colocamos un condicional utilizando la sentencia de control ***"if"*** y la función ***"predict"*** con argumento de entrada (***"[1, peso, color]"***). Si la función es igual a 1 significa que hablamos de un mango de consumo local, de caso contrario es un mango para importar. (Véase la Figura N° 82)

```
peso = float(entrada_peso.get())
color = float(entr_fruta.get())

if  pr.predict([1,peso,color]) == 1:
    mercad()
else:
    export()
```

Figura 82. Programación para clasificar mango (local y exportación)
Fuente: UNAC (2020); Autoría Propia.

Ejecutamos el programa, seleccionamos nuestro puerto de Arduino mediante el botón ***"CONECTAR PUERTO"***, encendemos la webcam con el botón ***"CONECTAR CAM"***, cargamos nuestra calibración de los parámetros HSV con el botón ***"ABRIR"***, colocamos el mango y podemos observar que automáticamente aparece el peso y finalmente hacemos click en predecir y observamos que nuestra neurona clasifica a dicho mango como para consumo local. (Véase la Figura N° 83)

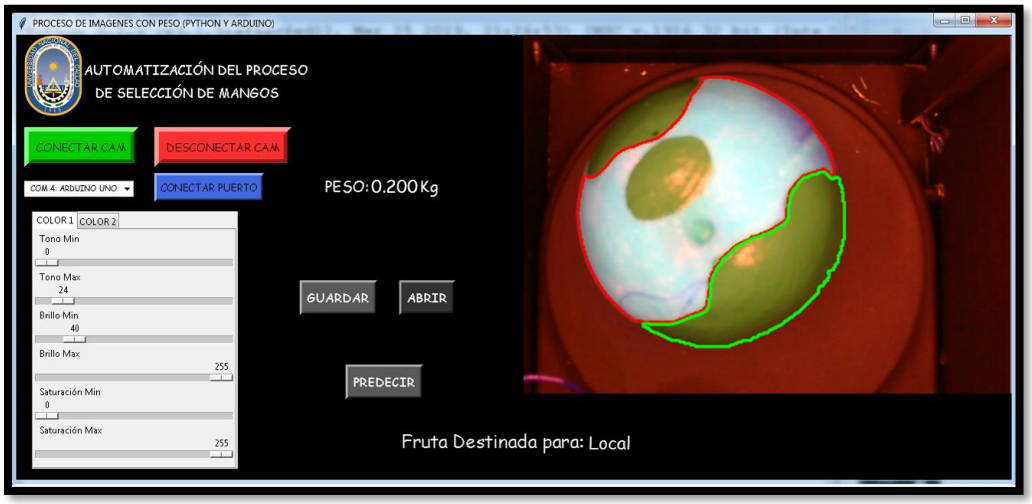


Figura 83. Sistema clasificador de mango aplicando PDI y RNA
Fuente: UNAC (2020); Autoría Propia.

- Implementación de una Red Neuronal Perceptron con Python

Al momento de implementar una red neuronal perceptron es necesario contar con grupo de datos para su entrenamiento, Se les solicito a los trabajadores de la hacienda Tambo Grande que nos trajeran 6 mangos y que de esos magos 3 de ellos sean para consumo local y los otros 3 sean para exportar. Una vez ya teniendo los mangos los sometimos a nuestro prototipo y creamos una tabla donde anotamos 2 tipos de variables (Peso y Color). (Véase la Tabla N°4, en la siguiente página)

TABLA 4. *TABLA DE CLASIFICACIÓN DE 6 MANGOS*

N°	Peso (gr) P1	Color P2	Clasificación
1	150	-0.3	Extranjero
2	90	0.5	Extranjero
3	210	0.2	Extranjero
4	20	-0.9	Local
5	40	-0.6	Local
6	30	-0.4	Local

Fuente: UNAC (2020); Autoría Propia.

Graficamos los puntos situados de la tabla como una serie de vectores (P1, P2) P1: Peso y P2: Color, con un círculo se graficará los mangos que son para el extranjero y con un aspa los mangos locales (Véase la Figura N° 84)

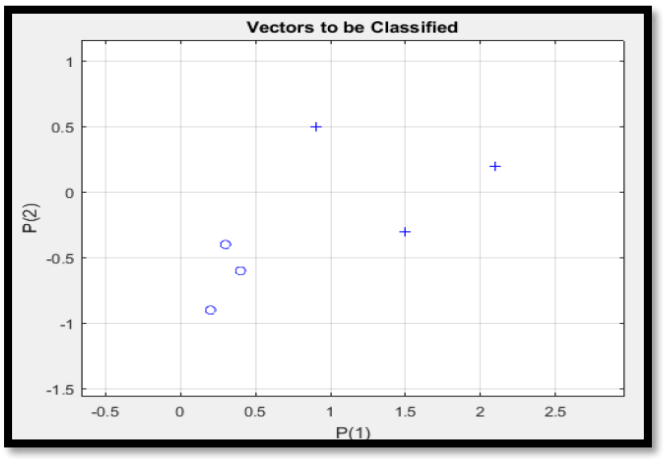


Figura 84. Grafica de vectores P1 y P2.

Fuente: UNAC (2020); Autoría Propia

Ahora pasaremos a trazar una recta que pueda dividir estos dos grupos, para eso nos apoyaremos en la teoría de una neurona perceptrón, una vez definido el método que vamos a utilizar procederemos a seleccionar nuestros pesos aleatorios: $[w^1]^T = [0.1, 0.1, 0.9]$ y un factor de aprendizaje $\alpha = 0.1$ y de la tabla N° 4.2. Vamos a codificar la palabra Extranjero como 1 y la palabra local como 0:

Iteración (k=1)

Entradas $x_1^1 = 1.5$, $x_2^1 = -0.3$. La salida deseada es $y_d^1 = 1$. La salida del combinador lineal s^1 cómo puede verse en la ecuación 36.

$$s^1 = 0.1(1) + 0.1(1.5) - 0.9(0.3) = -0.02 \quad \textbf{(36)}$$

$$y^1 = f(s^1) = 0$$

$$e^1 = y_d^1 - y^1 = 1 - 0 = 1$$

Actualizando el vector de pesos como se muestra en la ecuación 37

$$\begin{bmatrix} w_0^2 \\ w_1^2 \\ w_2^2 \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.9 \end{bmatrix} + 0.1(1)\frac{1}{1^2+(1.5)^2+(-0.3)^2}\begin{bmatrix} 1 \\ 1.5 \\ -0.3 \end{bmatrix} = \begin{bmatrix} 0.13 \\ 0.145 \\ 0.891 \end{bmatrix} \quad \textbf{(37)}$$

Iteración (k=2)

Entradas $x_1^2 = 0.9$, $x_2^2 = 0.5$. La salida deseada es $y_d^2 = 1$. La salida del combinador lineal s^2 es como puede verse en la ecuación 38

$$s^2 = 0.13(1) + 0.145(0.9) + 0.891(0.5) = 0.706 \quad \textbf{(38)}$$

$$y^1 = f(s^1) = 1$$

$$e^2 = y_d^2 - y^2 = 1 - 1 = 0$$

Notamos que aparentemente no existe error, pero falta corroborar con los demás casos de la Tabla N° 4.2.

Iteración (k=3)

Entradas $x_1^3 = 2.1$, $x_2^3 = 0.2$. La salida deseada es $y_d^3 = 1$. La salida del combinador lineal s^3 es como puede verse en la ecuación 39

$$s^3 = 0.13(1) + 0.145(2.1) + 0.891(0.2) = 0.6127 \quad \textbf{(39)}$$

$$y^1 = f(s^1) = 1$$

$$e^2 = y_d^2 - y^2 = 1 - 1 = 0$$

Iteración (k=4)

Entradas $x_1^4 = 0.2$, $x_2^4 = -0.9$. La salida deseada es $y_d^4 = 0$. La salida del combinador lineal s^4 es como puede verse en la ecuación 40.

$$s^4 = 0.13(1) + 0.145(0.2) + 0.891(-0.9) = -0.6429 \quad \textbf{(40)}$$

$$y^1 = f(s^1) = 0$$

$$e^2 = y_d^2 - y^2 = 0 - 0 = 0$$

Iteración (k=5)

Entradas $x_1^5 = 0.4$, $x_2^5 = -0.6$. La salida deseada es $y_d^5 = 0$. La salida del combinador lineal s^5 es como puede verse en la ecuación 41.

$$s^5 = 0.13(1) + 0.145(0.4) + 0.891(-0.6) = -0.3466 \quad \textbf{(41)}$$

$$y^1 = f(s^1) = 0$$

$$e^2 = y_d^2 - y^2 = 0 - 0 = 0$$

Iteración (k=6)

Entradas $x_1^6 = 0.3$, $x_2^6 = -0.4$. La salida deseada es $y_d^6 = 0$. La salida del combinador lineal s^6 es como puede verse en la ecuación 42.

$$s^6 = 0.13(1) + 0.145(0.3) + 0.891(-0.4) = -0.1829 \quad \textbf{(42)}$$

$$y^1 = f(s^1) = 0$$

$$e^2 = y_d^2 - y^2 = 0 - 0 = 0$$

Para terminar de corroborar la veracidad de estos pesos sinápticos volvemos a analizar la iteración (k=1) pero con estos nuevos pesos actualizados

Iteración (k=1)

Entradas $x_1^1 = 1.5$, $x_2^1 = -0.3$. La salida deseada es $y_d^1 = 1$. La salida del combinador lineal s^1 es como puede verse en la ecuación 43.

$$s^1 = 0.13(1) + 0.145(1.5) - 0.891(0.3) = 0.0802 \quad \textbf{(43)}$$

$$y^1 = f(s^1) = 1$$

$$e^1 = y_d^1 - y^1 = 1 - 1 = 0$$

Con esto queda demostrado cuales son los valores de los pesos sinápticos es como puede verse en la ecuación 44.

$$w = \begin{bmatrix} 0.13 \\ 0.145 \\ 0.891 \end{bmatrix} \quad \textbf{(44)}$$

Finalmente, la neurona perceptrón queda expresada a través de la ecuación de una recta. Como se observa en la ecuación 45.

$$s = 0.13 + 0.145 * P_1 + 0.891 * P_2 \quad \textbf{(45)}$$

De la Ecuación N° 4.10 se desprende lo siguiente:

$$y = \begin{cases} s > 0 \ \rightarrow y = 1; Extranjero \\ s \leq 0 \rightarrow y = 0; \qquad Local \end{cases}$$

Con lo cual se concluye lo siguiente si los parámetros P (1) y P (2) (Color y Peso) respectivamente son ingresados en la Ecuación N° 45 y si el valor de ***"s"*** es menor o igual a 0 significa que la neurona clasifico ese mango como un mango para mercado local y si ***"s"*** es mayor 0 significa que la neurona clasifico ese mango para mercado extranjero. Ahora pasamos a graficar la separabilidad lineal de la neurona ya entrenada. (Véase la Figura N° 85)

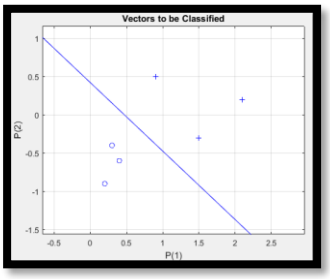


Figura 85. Separabilidad Lineal de la neurona ya entrenada.
Fuente: UNAC (2020); Autoría Propia.

4.2 Método de investigación

El método seleccionado es el método experimental, a continuación, veremos los pasos:

Paso 1: Observación del Fenómeno

Durante el mes de diciembre del 2018 pudimos observar que muchas veces los mismos trabajadores de la hacienda de Tambo Grande cometían varios errores al momento de clasificar los mangos, generando un descontento entre los clientes finales y pudimos observar que los trabajadores al momento de clasificar un mango lo hacen utilizando una balanza es decir lo clasifican por peso y también lo hacen mirando es decir que también lo clasifican por su color.

Paso 2: Hipótesis

El desarrollo de un algoritmo basado en la teoría de una neurona perceptrón con momentum cuyos parámetros de entrada sean el color (procesamiento de imágenes) y el peso (extracción paralela los valores de una balanza) nos dará como salida la automatización del proceso de selección de mangos y así evitar un descontento entre los clientes finales.

Paso 3: Experimentación

Actualmente ya hemos desarrollado un prototipo inicial con muy buenos resultados y ahora apuntamos a implementarlo de manera industrial con una faja, sensores y actuadores.

Paso 4: Conclusión

Se logró desarrollar un algoritmo basado en la teoría de una neurona perceptrón con momentum, teniendo 2 parámetros de entrada: el color de la futra (transformación de espacio vectorial RGB a HSV) y el peso (uso de un ADC de 24 bits y Arduino). Todo esto mediante el software libre Python

permitió automatizar el proceso de selección de magos y se logró solucionar el descontento entre los clientes finales

4.3 Población y muestra

La población que abarca la presente investigación son los mangos de la Huerta del compañero Javier Carrasco Jimenez. La presente investigación fue realiza en Diciembre del 2018 y el tamaño de la muestra fueron 6 magos de los cuales 3 son de consumo local y 3 son para consumo extranjero.

4.4 Lugar de estudio y periodo desarrollado

La huerta de mangos del Sr. Javier Carrasco Jimenez, ubicado en el distrito de Tambo grande, provincia de Piura departamento de Piura.

4.5 Técnicas e Instrumentos para la recolección de la información

- Una Balanza.

Instrumento que utilizamos para obtener el peso del mango, para luego ser utilizado como entrada numero 1 de la red neuronal percentron.

- Algoritmo reconocedor de colores

Desarrollado en PYTHON con ayuda de la librería OPENCV, para poder obtener los colores de la fruta (rojo amarillento y el verde amarillento) y que luego es escalonado desde -1 a 1 y con eso alimentar la segunda entrada de la red neuronal perceptron.

4.6 Análisis y procesamiento de datos

Se les solicito a los trabajadores de la hacienda Tambo Grande que nos trajeran 10 mangos y que de esos magos 5 de ellos sean para consumo local y los otros 5 sean para exportar y que luego vamos a poner a prueba con nuestro modelo de red neuronal perceptron. (Véase la Tabla N°5, en la siguiente página)

TABLA 5. *TABLA DE CLASIFICACIÓN DE 10 MANGOS PARA PROBAR NUESTRA RED NEURONAL*

N°	Peso (gr) P1	Color P2	Clasificación
1	140	-0.3	Extranjero
2	100	0.4	Extranjero
3	200	0.3	Extranjero
4	180	0.1	Extranjero
5	160	-0.2	Extranjero
6	45	-0.5	Local
7	50	-0.7	Local
8	35	-0.9	Local
9	45	-0.6	Local
10	55	-0.4	Local

Fuente: UNAC (2020); Autoría Propia.

Ahora ya teniendo estos datos vamos a poner a prueba la efectividad de nuestra red neuronal, cuyo modelo ya lo obtuvimos en la ecuación N°45

$$s = 0.13 + 0.145 * P_1 + 0.891 * P_2 \quad \textbf{(45)}$$

$$y = \begin{cases} s > 0 \ \rightarrow y = 1; Extranjero \\ s \leq 0 \rightarrow y = 0; \quad\quad Local \end{cases}$$

Prueba N° 1: Vamos a probar nuestro modelo de red neuronal con los valores: peso 140 y color -0.3. Como podemos observar en la ecuación N°46

$$s = 0.13 + 0.145 * (1.4) + 0.891 * (-0.3) \quad \textbf{(46)}$$

$$s = 0.0657$$

Como podemos observar el valor de s>0, por lo tanto, se clasifica el mango como un mango extranjero como se puede corroborar con la tabla N° 5.

Prueba N° 2: Vamos a probar nuestro modelo de red neuronal con los valores: peso 100 y color 0.4. Como podemos observar en la ecuación N°47

$$s = 0.13 + 0.145 * (1) + 0.891 * (0.4) \quad \textbf{(47)}$$

$$s = 0.6314$$

Como podemos observar el valor de s>0, por lo tanto, se clasifica el mango como un mango extranjero como se puede corroborar con la tabla N° 5.

Prueba N° 3: Vamos a probar nuestro modelo de red neuronal con los valores: peso 200 y color 0.3. Como podemos observar en la ecuación N°48

$$s = 0.13 + 0.145 * (2) + 0.891 * (0.3) \quad \textbf{(48)}$$

$$s = 0.6873$$

Como podemos observar el valor de s>0, por lo tanto, se clasifica el mango como un mango extranjero como se puede corroborar con la tabla N° 5.

Prueba N° 4: Vamos a probar nuestro modelo de red neuronal con los valores: peso 180 y color 0.1. Como podemos observar en la ecuación N°49

$$s = 0.13 + 0.145 * (1.8) + 0.891 * (0.1) \quad \textbf{(49)}$$

$$s = 0.4801$$

Como podemos observar el valor de s>0, por lo tanto, se clasifica el mango como un mango extranjero como se puede corroborar con la tabla N° 5.

Prueba N° 5: Vamos a probar nuestro modelo de red neuronal con los valores: peso 160 y color -0.2. Como podemos observar en la ecuación N°50

$$s = 0.13 + 0.145 * (1.6) + 0.891 * (-0.2) \quad \textbf{(50)}$$

$$s = 0.1838$$

Como podemos observar el valor de s>0, por lo tanto, se clasifica el mango como un mango extranjero como se puede corroborar con la tabla N° 5.

Prueba N° 6: Vamos a probar nuestro modelo de red neuronal con los valores: peso 45 y color -0.5. Como podemos observar en la ecuación N°51

$$s = 0.13 + 0.145 * (0.45) + 0.891 * (-0.5) \quad \textbf{(51)}$$

$$s = -0.25025$$

Como podemos observar el valor de s<0, por lo tanto, se clasifica el mango como un mango local como se puede corroborar con la tabla N° 5.

Prueba N° 7: Vamos a probar nuestro modelo de red neuronal con los valores: peso 50 y color -0.7. Como podemos observar en la ecuación N°52

$$s = 0.13 + 0.145 * (0.5) + 0.891 * (-0.7) \quad \textbf{(52)}$$

$$s = -0.4212$$

Como podemos observar el valor de s<0, por lo tanto, se clasifica el mango como un mango local como se puede corroborar con la tabla N° 5.

Prueba N° 8: Vamos a probar nuestro modelo de red neuronal con los valores: peso 35 y color -0.9. Como podemos observar en la ecuación N°53

$$s = 0.13 + 0.145 * (0.35) + 0.891 * (-0.9) \quad \textbf{(53)}$$

$$s = -0.62115$$

Como podemos observar el valor de s<0, por lo tanto, se clasifica el mango como un mango local como se puede corroborar con la tabla N° 5.

Prueba N° 9: Vamos a probar nuestro modelo de red neuronal con los valores: peso 45 y color -0.6. Como podemos observar en la ecuación N°54

$$s = 0.13 + 0.145 * (0.45) + 0.891 * (-0.6) \quad \textbf{(54)}$$

$$s = -0.33935$$

Como podemos observar el valor de s<0, por lo tanto, se clasifica el mango como un mango local como se puede corroborar con la tabla N° 5.

Prueba N° 10: Vamos a probar nuestro modelo de red neuronal con los valores: peso 55 y color -0.4. Como podemos observar en la ecuación N°55

$$s = 0.13 + 0.145 * (0.55) + 0.891 * (-0.4) \quad \textbf{(55)}$$

$$s = -0.14665$$

Como podemos observar el valor de s<0, por lo tanto, se clasifica el mango como un mango local como se puede corroborar con la tabla N° 5.

Agruparemos todos los resultados en la siguiente Tabla N°6

TABLA 6. *TABLA DE RESULTADOS PREELIMINARES DE LA RED NEURONAL*

N°	Peso (gr) P1	Color P2	Red Neuronal (s)
1	140	-0.3	0.0657
2	100	0.4	0.6314
3	200	0.3	0.6873
4	180	0.1	0.4801
5	160	-0.2	0.1838
6	45	-0.5	-0.25025
7	50	-0.7	-0.4212
8	35	-0.9	-0.62115
9	45	-0.6	-0.33935
10	55	-0.4	-0.14665

Fuente: UNAC (2020); Autoría Propia.

Ahora aplicamos la regla descrita en la Ecuación N° 45 para obtener la salida de la red neuronal y los resultados serán mostrados Tabla N° 7

TABLA 7. *TABLA DE RESULTADOS FINALES DE LA RED NEURONAL*

N°	Peso (gr) P1	Color P2	Red Neuronal (s)	Salida (Y)
1	140	-0.3	0.0657	1
2	100	0.4	0.6314	1
3	200	0.3	0.6873	1
4	180	0.1	0.4801	1
5	160	-0.2	0.1838	1
6	45	-0.5	-0.25025	0
7	50	-0.7	-0.4212	0
8	35	-0.9	-0.62115	0
9	45	-0.6	-0.33935	0
10	55	-0.4	-0.14665	0

Fuente: UNAC (2020); Autoría Propia.

Los mangos que tienen en su salida el numero 1 significan que son magos para el extranjero y los magos que tienen 0 en su salida son mangos de consumo local y ahora procederemos a mostrar los resultados comparativos entre la red neuronal y la clasificación realizada por los

trabajadores de la hacienda Tambo Grande. Como se puede observar en la Tabla N° 8

TABLA 8. *TABLA DE RESULTADOS COMPARATIVOS ENTRE LA RED NEURONAL Y LOS TRABAJADORES DE LA HACIENDA TAMBO GRANDE*

N°	Peso (gr) P1	Color P2	Red Neuronal	Trabajadores
1	140	-0.3	Extranjero	Extranjero
2	100	0.4	Extranjero	Extranjero
3	200	0.3	Extranjero	Extranjero
4	180	0.1	Extranjero	Extranjero
5	160	-0.2	Extranjero	Extranjero
6	45	-0.5	Local	Local
7	50	-0.7	Local	Local
8	35	-0.9	Local	Local
9	45	-0.6	Local	Local
10	55	-0.4	Local	Local

Fuente: UNAC (2020); Autoría Propia.

Como se puede observar en la Tabla N° 8 la efectividad que tiene nuestra red neuronal es del 100%, con lo cual se demuestra que nuestra red neuronal se encuentra bien entrenada y ya se encuentra lista para su implementación.

V Resultados

5.1 Resultados Descriptivos

Se le realizo pruebas al prototipo inicial con los mismos mangos que se utilizaron para entrenar la neurona (véase la Tabla N°4) y estos fueron los resultados de la balanza. (Véase la Tabla N°9)

TABLA 9. *TABLA COMPARATIVA DE RESULTADOS ENTRE BALANZA Y LA MAQUINA*

N°	Balanza	Sistema
1	1.5	0.148
2	0.9	0.9
3	2.1	2.11
4	0.2	0.2
5	0.4	0.4
6	0.3	0.29

Fuente: UNAC (2020); Autoría Propia.

Se le realizo pruebas al prototipo inicial con los mismos mangos que se utilizaron para entrenar la neurona (véase la Figura N° 86) y estos fueron los resultados del algoritmo de color. (Véase la Tabla N°10)

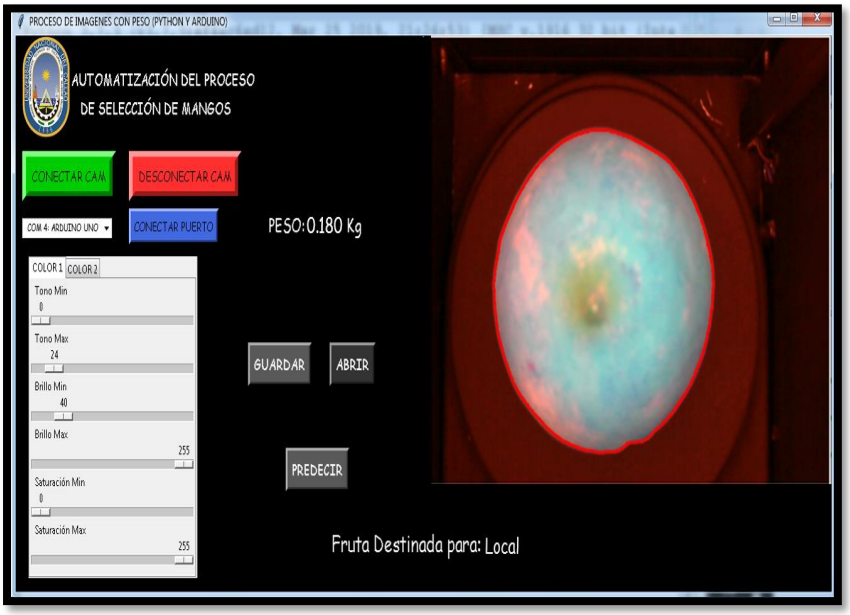


Figura 86. Automatización del proceso de selección de mangos
Fuente: UNAC (2020); Autoría Propia.

TABLA 10. *TABLA COMPARATIVA DE RESULTADOS ENTRE EL SENSOR DE COLOR Y LA MAQUINA*

N°	Sensor de Color	Sistema
1	-0.3	Verde
2	0.5	Verde
3	0.2	Verde
4	-0.9	Amarillo
5	-0.6	Amarillo
6	-0.4	Amarillo

Fuente: UNAC (2020); Autoría Propia.

Se le realizo pruebas al prototipo inicial con los mismos mangos que se utilizaron para entrenar la neurona y esto fueron los resultados. (Véase la Tabla N° 11)

TABLA 11. *TABLA COMPARATIVA DE RESULTADOS ENTRE EL TRABAJADOR Y LA MAQUINA*

N°	Peso /100 (P1)	Color (P2)	Trabajador	Maquina
1	1.5	-0.3	Extranjero	1
2	0.9	0.5	Extranjero	1
3	2.1	0.2	Extranjero	1
4	0.2	-0.9	Local	0
5	0.4	-0.6	Local	0
6	0.3	-0.4	Local	0

Fuente: UNAC (2020); Autoría Propia.

Con lo que se demuestra que la neurona perceptrón funciona correctamente, al igual que la balanza y el algoritmo de reconocimiento de colores con datos ya conocidos.

5.2 Resultados Inferenciales

Se le realizo pruebas al prototipo inicial con nuevos mangos que no pertenecen al grupo que se utilizó para entrenar a la neurona y estos fueron los resultados de la balanza. (Véase la Tabla N° 12)

***TABLA 12**. TABLA COMPARATIVA DE RESULTADOS ENTRE LA BALANZA Y LA MAQUINA*

°	Balanza	Sistema
1	0.8	0.81
2	1.2	1.19
3	2	2.01
4	0.3	0.29
5	0.5	0.51
6	0.4	0.42

Fuente: UNAC (2020); Autoría Propia.

Se le realizo pruebas al prototipo inicial con nuevos mangos que no pertenecen al grupo que se utilizó para entrenar a la neurona y estos fueron los resultados del algoritmo de color. (Véase la Tabla N° 13)

***TABLA 13.** TABLA COMPARATIVA DE RESULTADOS ENTRE EL SENSOR DE COLOR Y LA MAQUINA*

N°	Sensor de Color	Sistema
1	0.3	Verde
2	0.4	Verde
3	0.9	Verde
4	-0.7	Amarillo
5	-0.3	Amarillo
6	-0.8	Amarillo

Fuente: UNAC (2020); Autoría Propia.

Se utilizaron los mismos mangos de la Tabla N° 8 y 9 y con ayuda de los trabajadores se determinó lo siguiente: (Véase la Tabla N° 14)

***TABLA 14.** TABLA COMPARATIVA DE RESULTADOS ENTRE EL TRABAJADOR Y LA MAQUINA*

N°	Peso /100 (P1)	Color (P2)	Trabajador	Maquina
1	0.8	0.3	Extranjero	1
2	1.2	0.4	Extranjero	1
3	2	0.9	Extranjero	1
4	0.3	-0.7	Local	0
5	0.5	-0.3	Local	0
6	0.4	-0.8	Local	0

Fuente: UNAC (2019); Autoría Propia.

VI DISCUSIÓN DE RESULTADOS

6.1. Contrastación de la hipótesis con los resultados

Hipótesis Principal

“El desarrollo de un algoritmo basado en la teoría de redes neuronales cuyos parámetros de entrada sean el color (procesamiento de imágenes) y el peso (extracción paralela los valores de una balanza) permitirá la automatización del proceso de selección de mangos”

Demostración de la Hipótesis principal

Se desarrolló el proyecto en escala del equipo encargado de seleccionar los mangos, el cual cuenta con un interfaz gráfico en Python la cual se creó con la ayuda de las librerías de Tintar, Open y Py Serial, también mediante la librería Tkinter pudimos crear una gráfica y mediante la librería OpenCV y técnicas de procesamiento de imágenes aplicamos una transformación de espacio vectorial RGB a HSV con lo cual logramos hacer el seguimiento de un objeto a través del color y con la técnica de detección de bordes pudimos calcular las áreas y así determinar la predominancia de la fruta por lo tanto se logró desarrollar el algoritmo de selección automatizado. (Véase la Figura N° 87).

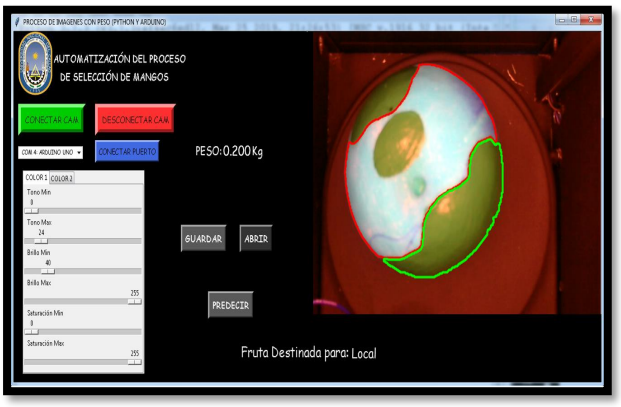


Figura 87. Demostración de la hipótesis principal
Fuente: UNAC (2020); Autoría Propia.

Por lo tanto, se demuestra que la hipótesis principal es válida, debido a que se logró demostrar que es factible desarrollar un algoritmo basado en la teoría de redes neuronales cuyos parámetros de entrada sean el color (procesamiento de imágenes) y el peso (extracción paralela los valores de una balanza), el cual nos permitió la automatización del proceso de selección de mangos.

Hipótesis Especifica N°1

“Es factible la implementación de un circuito electrónico utilizando el Módulo HX710B y el Arduino para extraer los valores de la Balanza, para luego importarlos a Python”.

Demostración de la Hipótesis N°1

Mediante el uso del módulo HX710B (ADC 24 bits), núcleo de ferrita y el Arduino Uno se pudo extraer los valores de una balanza de 5kg modelo SF-400 y mediante el uso de un LCD 16x2 y un módulo i2c se pudo visualizar los valores y luego estos se importaron a Python mediante el protocolo RS-232. (Véase la Figura N° 88).

Figura 88. Demostración de la hipótesis N° 1
Fuente: UNAC (2020); Autoría Propia.

Por lo tanto, se demuestra que la hipótesis N° 1 es válida, debido a que se logró demostrar que es factible la implementar un circuito electrónico utilizando el Módulo HX710B y el Arduino para extraer los valores de la Balanza, para luego importarlos a Python.

Demostración de la Hipótesis N°2

“Es viable la implementación de un prototipo de hardware a escala para la captura de imágenes con la finalidad de mantener constante el ambiente”.

Demostración de la Hipótesis N°2

Se implementó un prototipo hardware de captura de imágenes, el diseño se hizo en AutoCAD y luego se mandó a implementarlo, este prototipo contiene una webcam en la parte superior, una tira de leds color suave para poder mantener una iluminación constante, en la base del prototipo esta balanza y al costado esta un hardware especial para visualizar los datos de la balanza y se procedió a realizar una prueba con un celular. (Véase la Figura N° 89).

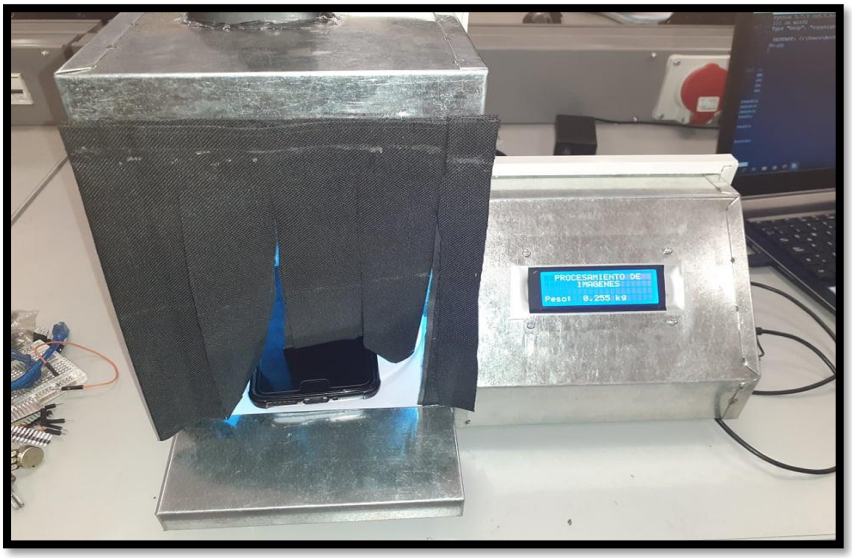


Figura 89. Demostración de la hipótesis N° 2
Fuente: UNAC (2020); Autoría Propia.

Por lo tanto, se demuestra que la hipótesis N° 2 es válida, debido a que se logró demostrar que es factible implementar un prototipo de hardware a escala para la captura de imágenes con la finalidad de mantener constante el ambiente

Hipótesis Especifica N°3

“Es viable el desarrollo de un código de programación en Python utilizando técnicas de reconocimiento de imágenes y técnicas de detección de bordes para reconocer el color de un mango”.

Demostración de la Hipótesis N°3

Se desarrolló una interfaz gráfica en Python con ayuda de las librerías de Tintar, Open y Py Serial, mediante la librería Tkinter podemos crear una gráfica, mediante la librería OpenCV y técnicas de procesamiento de imágenes aplicamos una transformación de espacio vectorial RGB a HSV con lo cual logramos hacer el seguimiento de un objeto a través del color y con la técnica de detección de bordes podemos calcular las áreas y así determinar la predominancia de la fruta y con la librería de Py Serial permite realizar una comunicación entre el Python y la Balanza. (Véase la Figura N° 90).

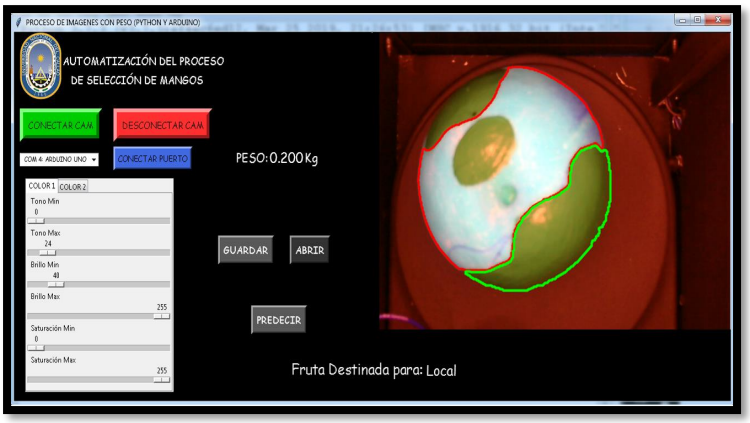


Figura 90. Demostración de la hipótesis N° 3
Fuente: UNAC (2020); Autoría Propia.

Por lo tanto, se demuestra que la hipótesis N° 3 es válida, debido a que se logró demostrar que es factible desarrollar un código de programación en Python utilizando técnicas de reconocimiento de imágenes y técnicas de detección de bordes para reconocer el color de un mango.

Hipótesis Especifica N°4

“Es factible crear una Neurona Perceptrón en Python para clasificar los mangos en 2 tipos (Local y Extranjero) cuyos valores de entrada de la neurona sean el color y el peso”.

Demostración de la Hipótesis N°4

Se procedió a crear una neurona Perceptrón con momentum con los datos de la Tabla N° 4.2 y el modelo matemático de la Figura N° 10, se procedió a realizar el entrenamiento con la ecuación N° 3 y se obtuvieron los pesos sinápticos mostrados en la ecuación N° 44 y finalmente la neurona queda de la siguiente manera. (Véase la Figura N° 91).

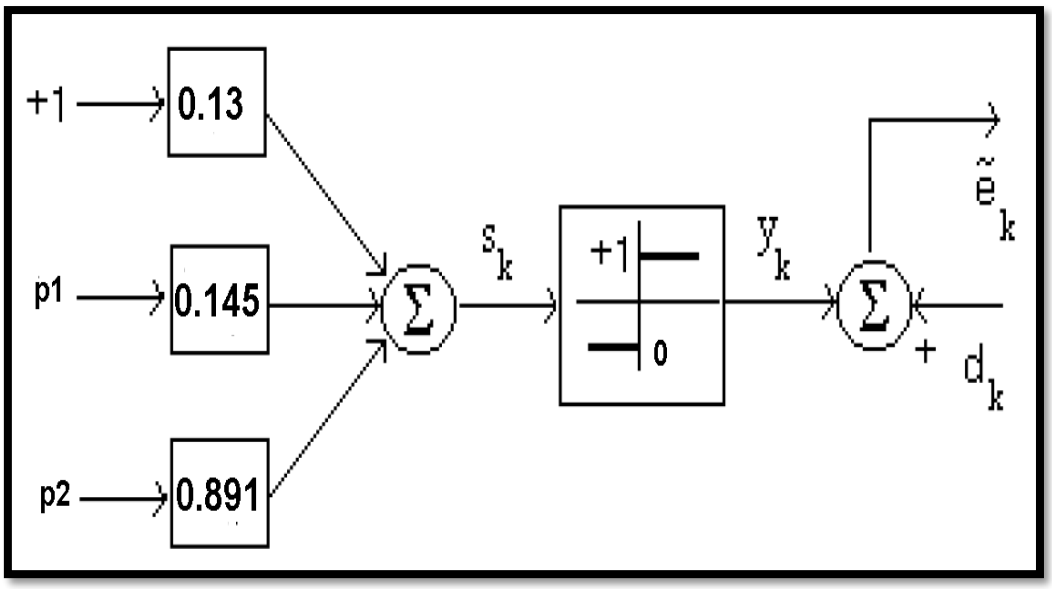


Figura 91. Demostración de la hipótesis N° 4
Fuente: UNAC (2020); Autoría Propia.

Por lo tanto, se demuestra que la hipótesis N° 4 es válida, debido a que se logró demostrar que es factible crear una Neurona Perceptrón en Python para clasificar los mangos en 2 tipos (Local y Extranjero) cuyos valores de entrada de la neurona sean el color y el peso.

6.2. Contrastación de los resultados con otros estudios similares

Ya existe un trabajo realizado por Franzua Oblitas Aristondo. (Véase la Figura N° 92), el trabajo consiste en una aplicación realizada en Python, aplicando técnicas de procesamiento de imágenes para seleccionar el color (transformación de espacio vectorial RGB a HSV) y técnicas de detección de bordes para calcular el área del mango y así determinar el color predominante en el mango y con eso poder clasificarlo. La ventaja que tiene nuestro sistema es que para clasificar el mango lo hace a través de una neurona cuyos datos de entrada son color y peso y te da la opción de poder ingresar más información y así realizar una clasificación más exacta y a diferencia del caso anterior en el de nosotros se ha acondicionado un espacio para poder mantener una iluminación constante y así poder tener un mejor procesamiento de imágenes.

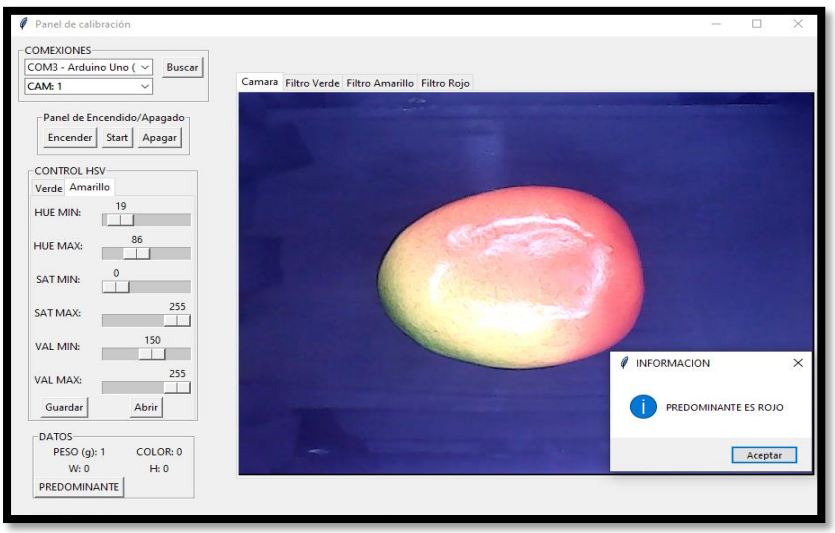


Figura 92. Automatización del proceso de selección de mango aplicando PDI.
Fuente: UNAC (2020); Autoría Propia.

CONCLUSIONES

- Se logro desarrollar el algoritmo en Python basado en la teoría de redes neuronales cuyos parámetros de entrada son el color (procesamiento de imágenes) y el peso (extracción paralela los valores de una balanza), con eso se logró automatizar el proceso de selección de mangos, por lo tanto, se da como válida la hipótesis principal planteada.
- Se logró implementar un circuito electrónico mediante un módulo HX710B (ADC de 24 bits), un núcleo de ferrita, utilizando un Arduino UNO, un módulo i2c y un LCD 16x2 para extraer los valores de una balanza SF-400, para luego importarlos al Python, por lo tanto, se da como válida la primera hipótesis especifica planteada.
- Se logró diseñar en AutoCAD e implementar un prototipo de hardware a escala para la captura de imágenes que cuenta con una tira de leds color suave para poder mantener el ambiente constante y mejorar la etapa de procesamiento de imágenes, por lo tanto, se da como válida la segunda hipótesis especifica planteada.
- Se logró desarrollar una interfaz gráfica en Python con la ayuda de las librerías (OpenCV, Tkinter y Py Serial) para reconocer el color de una fruta aplicando las técnicas de procesamiento de imágenes: Transformación de espacio vectorial RGB a HSV para poder determinar los colores que están presentes en el mango y detección de bordes para saber el área de los colores y así determinar la predominancia del mango, por lo tanto, se da como válida la tercera hipótesis especifica planteada.
- Se logró desarrollar una Neurona Perceptrón con Momentum en Python para clasificar los mangos en 2 tipos (Local y Extranjero) cuyos valores de entrada son el: Color (Procesamiento de Imágenes) y el peso (Balanza electrónica y Arduino), por lo tanto, se da como válida la cuarta hipótesis especifica planteada.

RECOMENDACIONES

- Al momento de utilizar la aplicación del prototipo de automatización del proceso de selección de mangos tener instalado en su computadora o laptop el Python versión 3.7 para adelante ya que de la versión 3.7 para abajo algunas líneas de programación cambian, adicionalmente tener instalado las siguientes librerías: OpenCV, Tkinter y Py Serial.
- Siempre encender primero la webcam, luego cargar los valores de los parámetros HSV y finalmente conectarse al puerto serie del Arduino.
- Siempre que se haga una modificación entre los parámetros HSV utilizar el botón de guardar para salvar la última calibración.

REFERENCIAS BIBLIOGRÁFICAS

"Red Neuronal de Topología Flexible". **Aguilar, R. 1999.** Bolivia : s.n., 1999.

Aceves, Dr. Alejandro. 2017. *Seminario del proyecto de investigación en robótica humanoide .* Mexico : Tecnologico de Monterrey, 2017.

Aguilera, Diego González. 2015. *PROCESAMIENTO DE IMÁGENES .* s.l. : UNIVERSIDAD DE SALAMANCA , 2015.

EcuRed. 2018. EcuRed. *EcuRed.* [En línea] Setiembre de 2018. [Citado el: 16 de Noviembre de 2019.] https://www.ecured.cu/Modelo_HSV.

HackeandoTec. 2016. *Clasificacion de Manzanas y Piñas utilizando Redes Neuronales.* HackeandoTec, 2016.

HBM. 2017. [En línea] 2017. https://www.hbm.com/es/7163/el-puente-de-wheatstone-galgas-extensometricas/.

K., Alexander Mordvintsev y Abid. 2013. Tutorial OpenCV- Python. [En línea] 2013. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html#morphological-ops.

—. 2013. Tutoriales de OpenCV - Python. [En línea] 2013. [Citado el: 20 de Marzo de 2019.] https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_colorspaces/py_colorspaces.html#converting-colorspaces.

Oblitas, Juan Salcedo y Franzua. 2018. *Diseño e Implementacion de un sistema de control de balanzas para optimizar el rebose en los contenedores de la empresa Mondelez.* Lima : UNAC, 2018.

OMEGA. 2019. [En línea] 2019. https://es.omega.com/prodinfo/celulas-de-carga.html.

Pajuelo, Dominick. 2009. Proyecto Electronico . [En línea] 2009. https://www.proyectoelectronico.com/arduino/bascula-con-arduino-hx710.html.

Semiconductor, Avia. 2017. All Datasheet. *All Datasheet.* [En línea] 8 de Junio de 2017. [Citado el: 25 de Febrero de 2019.] file:///C:/Users/pc001n/Downloads/HX710B-RoHS%20(2).pdf.

Sphinx. 2017. Sphinx. *Sphinx.* [En línea] Julio de 2017. [Citado el: 15 de Noviembre de 2019.] http://docs.python.org.ar/tutorial/3/real-index.html.

stackoverflow. 2017. stackoverflow . *stackoverflow .* [En línea] 18 de Julio de 2017. [Citado el: 13 de Noviembre de 2019.] https://stackoverflow.com/questions/44480131/python-opencv-hsv-range-finder-creating-trackbars.

UNAC. 2018. *Diseño e Implementación de un Sistema de Control de Balanzas Industriales para Optimizar el Rebose de los Contenedores de la Empresa Mondelez.* Callao : UNAC, 2018.

Unipython. 2016. Unipython . *Unipython .* [En línea] 28 de Julio de 2016. [Citado el: 15 de Noviembre de 2019.] https://unipython.com/tkinter-introduccion/.

Valdivieso, Pedro A. Castillo. 2017. *Redes neuronales. El perceptrón.* s.l. : Universidad de Granada, 2017.

ANEXOS

➢ Matriz de consistencia

Título: "AUTOMATIZACIÓN DEL PROCESO DE SELECCIÓN DE MANGOS UTILIZANDO PROCESAMIENTO DE IMÁGENES Y REDES NEURONALES"

PROBLEMAS	OBJETIVOS	HIPOTESIS	VARIABLES	METODOLOGIA
Problema General: ¿De qué manera podemos automatizar el proceso de selección de mangos utilizando procesamiento de imágenes y redes neuronales?	**Objetivo General:** Desarrollar un algoritmo de control que permita automatizar la clasificación de los mangos en 2 tipos (Mercado Local y Mercado Extranjero) mediante la aplicación de una Neurona Perceptrón y utilizando como señales de entrada: el color y peso.	**Hipótesis General:** "El desarrollo de un algoritmo basado en la teoría de redes neuronales cuyos parámetros de entrada sean el color (procesamiento de imágenes) y el peso (extracción paralela los valores de una balanza) permitirá la automatización del proceso de selección de mangos"	**Variable Independiente:** ❖ Algoritmo de Control basado en procesamiento de imágenes y redes neuronales. **Indicador:** • Algoritmo de control diseñado para automatizar el proceso de selección de mangos. **Variables Dependientes** ❖ El proceso de Selección de Mangos **Indicador:** • Proceso actual realizado por trabajadores ❖ Optimización de los tiempos de producción **Indicador:** • Tiempo que se demora en clasificar un mango utilizando el algoritmo	**Analítico** Para realizar un análisis de los elementos que conforman este proyecto el método de investigación a utilizar es el Método Analítico. **Experimental** Porque se realizaron diferentes pruebas preliminares que nos permitirá desarrollar el algoritmo de control para la automatización del proceso de selección de mangos . **Temporal** Porque el estudio está circunscrito a un intervalo de tiempo Enero 2020- Agosto 2020. **Espacial** Se realizar la investigación tomando como referencia la huerta de mangos del Sr. Javier Carrasco Jimenez
Problema Especifico N°1 ¿Cómo podemos importar los valores de una balanza electrónica a Python?	**Objetivo Especifico N°1:** Implementar un circuito electrónico mediante un Arduino y un Módulo HX710B para extraer los datos de una balanza e importarlos a Python	**Hipótesis Especifica N°1:** "Es factible la implementación de un circuito utilizando el HX710B y un Arduino para extraer los valores de una balanza"		
Problema Especifico N°2 ¿De qué manera podemos mantener constante el ambiente para la captura de imágenes?	**Objetivo Especifico N°2:** Implementar un prototipo de hardware a escala para la captura de imágenes y así poder mantener constante el ambiente	**Hipótesis Especifica N°2:** "Es viable la implementación de un prototipo de hardware para la captura de imágenes con la finalidad de mantener constante el ambiente"		
Problema Especifico N°3 ¿De qué manera podemos reconocer el color de un mago e importarlos a Python?	**Objetivo Especifico N°3:** Desarrollar un código de programación en Python para reconocer los colores de un mango utilizando procesamiento de imágenes.	**Hipótesis Especifica N°3:** "Es viable el desarrollo de un código de en Python utilizando técnicas de reconocimiento de imágenes y detección de bordes para reconocer el color de un mango"		
Problema Especifico N°4 ¿De qué manera podemos utilizar los parámetros (color y peso) para clasificar los mangos?	**Objetivo Especifico N°4:** Crear una Neurona Perceptrón en Python para clasificar los mangos en 2 tipos: local y extranjero y cuyos valores de entrada son el color y el peso.	**Hipótesis Especifica N°4:** "Es factible la creación de una neurona para clasificar los mangos valiéndose de valores de entrada como Color y Peso"		

- **Instrumentos validados**

Para nuestra presente investigación utilizamos como herramienta de recolección de datos una balanza y algoritmo de reconocimiento de colores por lo no requerimos adjuntar una validación de instrumento.

- **Programación del Arduino:**

```
#include "Arduino.h"
#include <LiquidCrystal_I2C.h>
#define DOUT 2
#define PD_SCK 3
#define factor 0.00959
//Crear el objeto lcd  dirección  0x3F y 20 columnas x 4 filas
LiquidCrystal_I2C lcd(0x3F,20,4);

unsigned long value;
long weight;

long get_weight()
{
   digitalWrite(PD_SCK, LOW);
   delayMicroseconds(1);
   // wait for the chip to become ready:
   while (digitalRead(DOUT) == HIGH);
   value = 0;
  for (int i = 23; i > -1; i--){ //bitWrite23 =bit24
   digitalWrite(PD_SCK, HIGH);
   delayMicroseconds(1);
   digitalWrite(PD_SCK, LOW);
   if (digitalRead(DOUT) == HIGH){bitSet(value, i);}
   }
    // para que siga leyendo muestras a 10Hz:
    digitalWrite(PD_SCK, HIGH);
    delayMicroseconds(1);
    digitalWrite(PD_SCK, LOW);
    delayMicroseconds(1);

 return value; // todos 1 = 1677215
}

void setup() {
   Serial.begin(9600);
   pinMode(DOUT, INPUT);
   pinMode(PD_SCK, OUTPUT);
```

```
   lcd.init();
   lcd.backlight();
   lcd.setCursor(0,3);
   lcd.print("PROCESAMIENTO DE ");
   lcd.setCursor(1,8);
   lcd.print("IMAGENES");

}

void loop() {
  weight = get_weight();
  float redondeado;
  redondeado = round(weight*factor);
  Serial.println(redondeado,0);
  lcd.setCursor(3,0);
  lcd.print("Peso: ");
   lcd.print(redondeado,3);//3 decimales
  lcd.print("kg");

}
```

- **Programa Python**

```
import tkinter as tk
from tkinter import ttk
from tkinter import*
from PIL import Image, ImageTk
import numpy as np
from tkinter import filedialog
from tkinter import messagebox
import serial
import sys,glob
import random
import threading
import cv2
import time
global camara



camara = cv2.VideoCapture(0)

ventana = tk.Tk()
ventana .title("PROCESO DE IMAGENES CON PESO (PYTHON Y
ARDUINO)")
ventana .config(bg = "Black")
```

```
ventana .geometry('1300x580')
miframe=Frame(ventana).pack()
texto =Label(miframe, text="AUTOMATIZACIÓN DEL PROCESO",bg ="black",fg = "white",font =("Comic Sans MS",13)).place(x=85,y=30)
texto1 =Label(miframe, text="DE SELECCIÓN DE MANGOS",bg ="black",fg = "white", font =("Comic Sans MS",13)).place(x=100,y=60)
ventana_camara = tk.Label(ventana)
ventana_camara.pack(anchor=tk.SE)

imagen = PhotoImage(file="unac.png")
Label (ventana, image = imagen, bd =0).place(x=10,y=0)


#.....................LISTAS DE ARDUINO.............#
#.....................LISTAS DE ARDUINO.............#

available = []

for i in range(256):
    try:
        s = serial.Serial('COM'+str(i),9600)
        time.sleep(2)
        available.append( (s.portstr))

    except serial.SerialException:
        pass

listas_arduino = ttk.Combobox(ventana,width=20, state="readonly")
listas_arduino.place(x=10, y=190)


def listaArduino():
    for s in available:
        #print ("%s" % (s))
        listas_arduino['values'] =(s)
        listas_arduino.set(s)
        if s!=" ":
            sensor.after(500, sensor_tem)


def sensor_tem():
    dato = s.readline()
    senso = dato[0:5]
    readingt.set(senso)
    sensor.after(50, sensor_tem)
```

```
readingt = StringVar()

sensor = Label(ventana, textvariable = readingt,bg ="gray65", font =("Comic Sans MS",15))
sensor.place(x= 470, y = 180)
text =Label(miframe, text="PESO: ",bg ="black",fg = "white", font =("Comic Sans MS",15)).place(x=400,y=180)
tex =Label(miframe, text="Kg",bg ="black",fg = "white", font =("Comic Sans MS",15)).place(x=525,y=180)

buscar_arduino = Button(miframe, text ="CONECTAR PUERTO", relief="raised", borderwidth=5,command = listaArduino, bg ="royal blue",fg = "black", font =("Comic Sans MS",10)).place(x=180,y=180)


#.................................camara.....................

def conectar2():
    global camara
    if camara.isOpened() == True:
        _,frame = camara.read()
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        color_real = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        # .....................PRIMER COLOR.........#
        tmin=w1_1.get()
        tmax=w2_1.get()
        bmin=w3_1.get()
        bmax=w4_1.get()
        smin=w5_1.get()
        smax=w6_1.get()
        minimo_1= np.array([tmin,smin,bmin])
        maximo_1= np.array([tmax,smax,bmax])
        mascara = cv2.inRange(hsv,minimo_1,maximo_1)
        kernel_1= np.ones((3,3),np.uint8)
        erosion_1= cv2.erode(mascara,kernel_1,iterations = 3)
        dilation_1 = cv2.dilate(erosion_1,kernel_1,iterations = 2)

        #.............SEGUNDO COLOR.............#
        tomin=w1_2.get()
        tomax=w2_2.get()
        brmin=w3_2.get()
        brmax=w4_2.get()
        samin=w5_2.get()
        samax=w6_2.get()
        minimo_2= np.array([tomin,samin,brmin])
        maximo_2= np.array([tomax,samax,brmax])
```

```
        mascara_2 = cv2.inRange(hsv,minimo_2,maximo_2)
        kernel_2= np.ones((3,3),np.uint8)
        erosion_2= cv2.erode(mascara_2,kernel_2,iterations = 3)
        dilation_2 = cv2.dilate(erosion_2,kernel_2,iterations = 2)

        #..................... CONTORNOS Y AREAS DE LOS VALORES.....#
        contornos,_   =   cv2.findContours(dilation_1,cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
        contorn,__    =    cv2.findContours(dilation_2,cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

        fruta_madura = 0
        fruta_verde = 0

        for c in contornos:
            area = cv2.contourArea(c)
            fruta_madura = area
            if area > 13000 and area < 200000:
                cv2.drawContours(frame,contornos,0,(255,255,0),2,
cv2.LINE_AA)
            continue
        for a in contorn:
            area_1 = cv2.contourArea(a)
            fruta_verde = area_1
            if area_1 > 13000 and area_1 < 200000:
                cv2.drawContours(frame,contorn,0,(0,255,0),2, cv2.LINE_AA)
            continue

        if fruta_madura > fruta_verde:
            color.set(1)
        if fruta_madura < fruta_verde:
            color.set(0)

        img = Image.fromarray(frame)
        imgtk = ImageTk.PhotoImage(image = img)
        ventana_camara.imgtk = imgtk
        ventana_camara.configure(image=imgtk)
        ventana_camara.after(10,conectar2)



    elif camara.isOpened() == False:
        camara = cv2.VideoCapture(0)

    else:
        None
```

```
color = StringVar()
entr_fruta = Entry(ventana, textvariable= color ,bg ="light grey", font =("Comic Sans MS",15))
entr_fruta.pack()
entr_fruta.place(x = -700, y = -5500)



#.....................#
tex_expor =Label(miframe, text="Exportación",bg ="black",fg = "white", font =("Comic Sans MS",18))
tex_expor.pack()
tex_expor.place(x=-600,y=-500)

tex_mercad =Label(miframe, text="Consumo Local",bg ="black",fg = "white", font =("Comic Sans MS",18))
tex_mercad.pack()
tex_mercad.place(x=-600,y=-900)

tex_ =Label(miframe, text="Fruta Destinada para:",bg ="black",fg = "white", font =("Comic Sans MS",18))
tex_.pack()
tex_.place(x=500,y=510)

def export():
    tex_expor.place(x=780,y=510)
    tex_mercad.place(x=-600,y=-900)

def mercad():
    tex_mercad.place(x=750,y=510)
    tex_expor.place(x=-600,y=-500)
    #.............NEURONA PERCEPTRON.............#
def neurona_perceptron():
    class Perceptron:
        def __init__(self,input_number,step_size = 0.1):
            self._ins = input_number
            self._w = [random.random() for _ in  range(input_number)]
            self._eta = step_size

        def predict(self,inputs):
            weighted_average = sum(w*elm for w,elm in zip (self._w,inputs))
            if weighted_average > 0:
                return 1
            return 0
        def train (self,inputs,ex_output):
```

```
            output = self.predict(inputs)
            error = ex_output - output
            if error !=0:
                self._w = [w + self._eta*error*x for w,x in
                zip (self._w,inputs)]
            return error

    input_data = [[1.5,-0.3,1], #PESO, COLOR , GRUPO
                  [0.9,0.5,1],
                  [2.1,0.2,1],
                  [0.2,-0.9,0],
                  [0.4,-0.6,0],
                  [0.3,-0.4,0]]


    pr = Perceptron(3)

    for _ in range (15000):
        for  person in input_data:
            output = person[-1]
            inp = [1] + person[0:-1]
            err = pr.train(inp,output)

    peso = float(entrada_peso.get())
    color = float(entr_fruta.get())

    if  pr.predict([1,peso,color]) == 1:
        export()
    else:
        mercad()



#.........PREDECIR.........#

predecir = Button(text ="PREDECIR", relief="raised", borderwidth=5, bg
="gray35",fg = "white", command = neurona_perceptron, font =("Comic
Sans MS",12)).place(x=430,y=430)


#..... CONECTAR CAMARA .....
def camara1():
    conectar2()
    ventana_camara.pack(anchor=tk.SE)
```

```
conectar = Button(miframe, text ="CONECTAR CAM", relief="raised", borderwidth=11, command = camara1,bg ="green3",fg = "black", font =("Comic Sans MS",11)).place(x=10,y=120)

#..................... DESCONECTAR CAMARA.....
def camara_des():
    camara.release()
    ventana_camara.place(x=645,y=-600)

desconectar = Button(miframe, text ="DESCONECTAR CAM", relief="raised", borderwidth=11 , command = camara_des ,bg ="firebrick1",fg = "black", font =("Comic Sans MS",11)).place(x=180,y=120)

#.............SLIDERS.............#
notebook = ttk.Notebook(ventana)
notebook.place(x=20 ,y=230)
pes1 = ttk.Frame(notebook,height=310,width=265)
pes2 = ttk.Frame(notebook,height=310,width=265)
notebook.add(pes1, text='COLOR 1')
notebook.add(pes2, text='COLOR 2')

#.............SLIDER 1...
w1_1 = tk.Scale(pes1, label='Tono Min',from_=0, to=255,
           length=260,width=9,orient=HORIZONTAL)
w1_1.place(x=0,y=0)

w2_1 = tk.Scale(pes1, label='Tono Max',from_=0, to=255,
           length=260,width=9,orient=HORIZONTAL)
w2_1.place(x=0,y=50)

w3_1= tk.Scale(pes1, label='Brillo Min',from_=0, to=255,
           length=260,width=9,orient=HORIZONTAL)
w3_1.place(x=00,y=100)

w4_1 = tk.Scale(pes1, label='Brillo Max',from_=0, to=255,
           length=260,width=9,orient=HORIZONTAL)
w4_1.place(x=0,y=150)

w5_1 = tk.Scale(pes1, label='Saturación Min',from_=0, to=255,
           length=260,width=9,orient=HORIZONTAL)
w5_1.place(x=0,y=200)

w6_1 = tk.Scale(pes1, label='Saturación Max',from_=0, to=255,
          length=260,width=9,orient=HORIZONTAL)
w6_1.place(x=0,y=250)
```

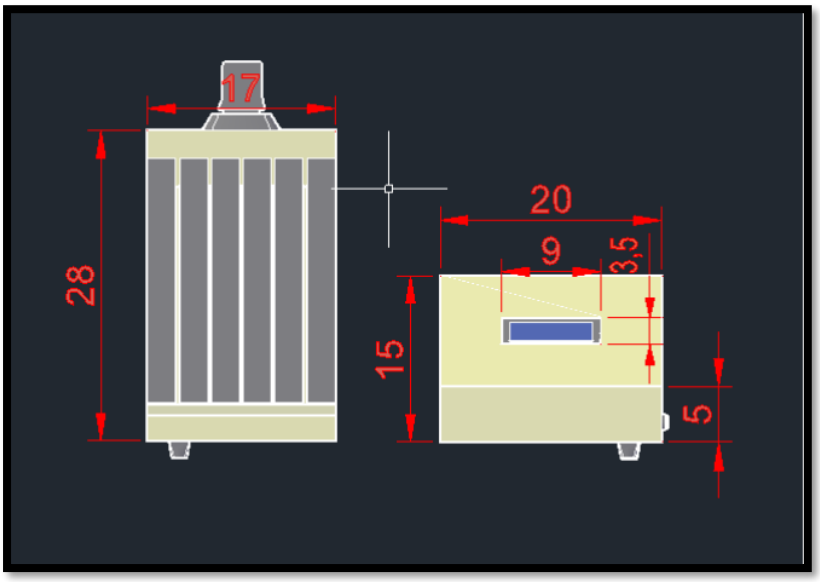
```
#.............SLIDER 2...
w1_2 = tk.Scale(pes2, label='Tono Min',from_=0, to=255,
           length=260,width=9,orient=HORIZONTAL)
w1_2.place(x=0,y=0)

w2_2 = tk.Scale(pes2, label='Tono Max',from_=0, to=255,
           length=260,width=9,orient=HORIZONTAL)
w2_2.place(x=0,y=50)

w3_2= tk.Scale(pes2, label='Brillo Min',from_=0, to=255,
           length=260,width=9,orient=HORIZONTAL)
w3_2.place(x=00,y=100)

w4_2 = tk.Scale(pes2, label='Brillo Max',from_=0, to=255,
           length=260,width=9,orient=HORIZONTAL)
w4_2.place(x=0,y=150)

w5_2 = tk.Scale(pes2, label='Saturación Min',from_=0, to=255,
           length=260,width=9,orient=HORIZONTAL)
w5_2.place(x=0,y=200)

w6_2 = tk.Scale(pes2, label='Saturación Max',from_=0, to=255,
          length=260,width=9,orient=HORIZONTAL)
w6_2.place(x=0,y=250)

#.................GUARDAR EN .TXT ..............................#

def guardarHSV():
   archivo = filedialog.asksaveasfilename(initialdir = "/", title = "Guardar
como",filetypes = (("Documentos de texto","*.txt"),("todos los
archivos","*.*")), defaultextension=".txt")
   if archivo!='':
      archivo = open(archivo, 'w')

      tmin=w1_1.get()
      tmax=w2_1.get()
      bmin=w3_1.get()
      bmax=w4_1.get()
      smin=w5_1.get()
      smax=w6_1.get()

      tomin=w1_2.get()
      tomax=w2_2.get()
      brmin=w3_2.get()
      brmax=w4_2.get()
      samin=w5_2.get()
```

```
        samax=w6_2.get()
        variable_string ="{}" "\n" "{}" "\n" "{}" "\n" "{}" "\n" "{}" "\n" "{}" "\n"  "{}"
"\n"       "{}"       "\n"       "{}"       "\n"       "{}"       "\n"       "{}"       "\n"       "{}"
"\n".format(tmin,tmax,bmin,bmax,smin,smax,tomin,tomax,brmin,brmax,sa
min,samax)
        archivo.write(variable_string)
        archivo.close()
        messagebox.showinfo("ATENTO !! ","Tus valores del HSV fueron
guardados")

guardar_txt = Button(miframe, text ="GUARDAR", relief="raised",
borderwidth=5,
                bg ="gray35",fg = "white", command = guardarHSV,
                font =("Comic Sans MS",12)).place(x=370,y=320)

#..................... MOSTRAR EN OTRA VENTANA .TXT .....

def mostrar():
    archivo = filedialog.askopenfilename(initialdir = "/", title = "Seleccione
archivo",
    filetypes = (("Documentos de texto","*.txt"),("todos los
archivos","*.*")),defaultextension=".txt")
    if archivo!='':
        archivo_ = open(archivo, 'r')
        contenido = []

        for linea in archivo_.readlines():
            contenido.append(linea.split())

        w1_1.set(contenido[0][0])
        w2_1.set(contenido[1][0])
        w3_1.set(contenido[2][0])
        w4_1.set(contenido[3][0])
        w5_1.set(contenido[4][0])
        w6_1.set(contenido[5][0])
        w1_2.set(contenido[6][0])
        w2_2.set(contenido[7][0])
        w3_2.set(contenido[8][0])
        w4_2.set(contenido[9][0])
        w5_2.set(contenido[10][0])
        w6_2.set(contenido[11][0])


ajustar_hsv = Button(miframe, text ="ABRIR", relief="raised",
borderwidth=5,
              bg ="gray20",fg = "white", command = mostrar,
```

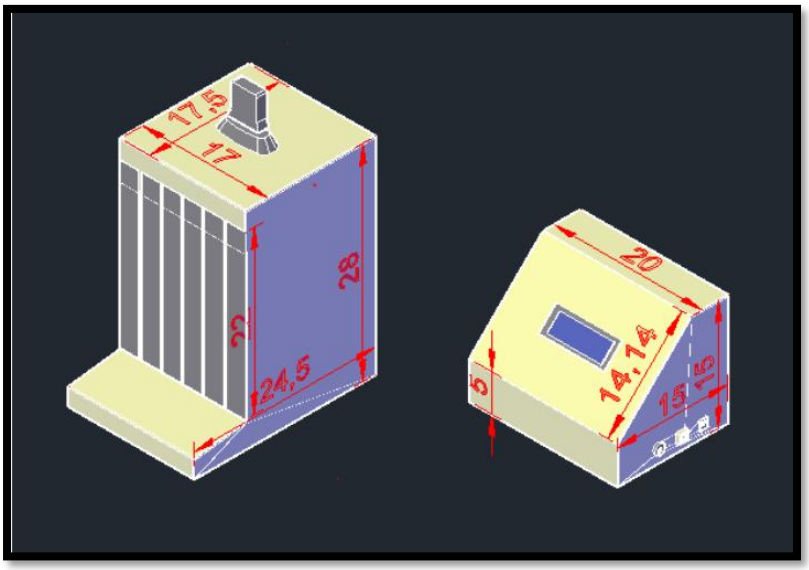
```
                font =("Comic Sans MS",12)).place(x=500,y=320)
#.............
ventana.mainloop()
```

- **Dibujo Mecánico en AutoCAD**

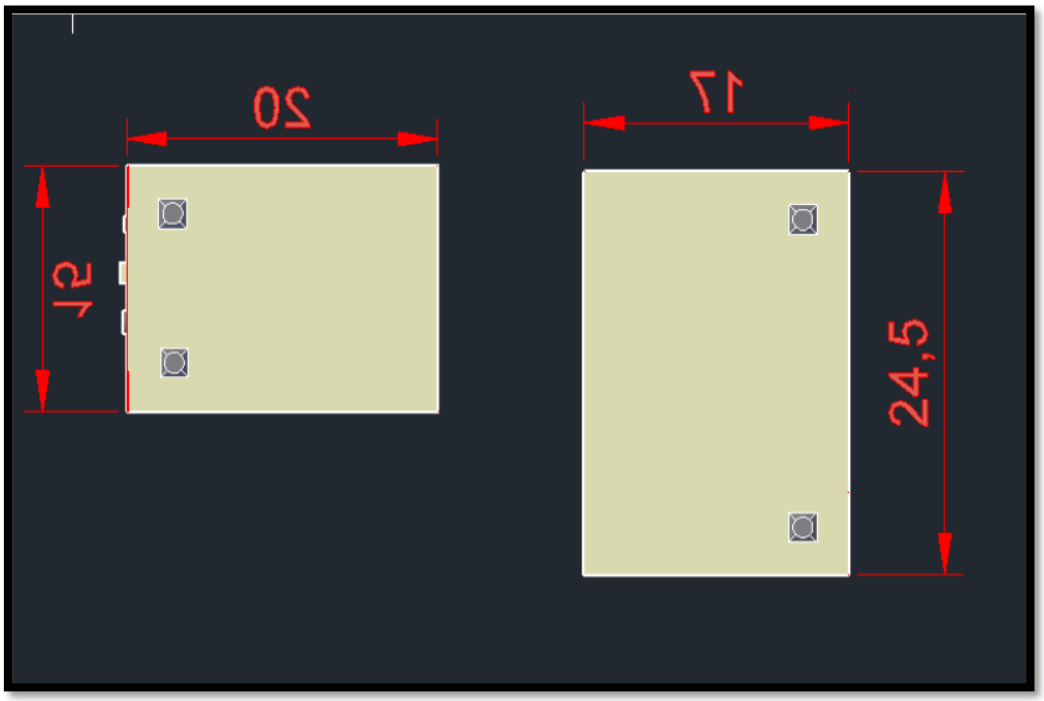
VISTA FRONTAL



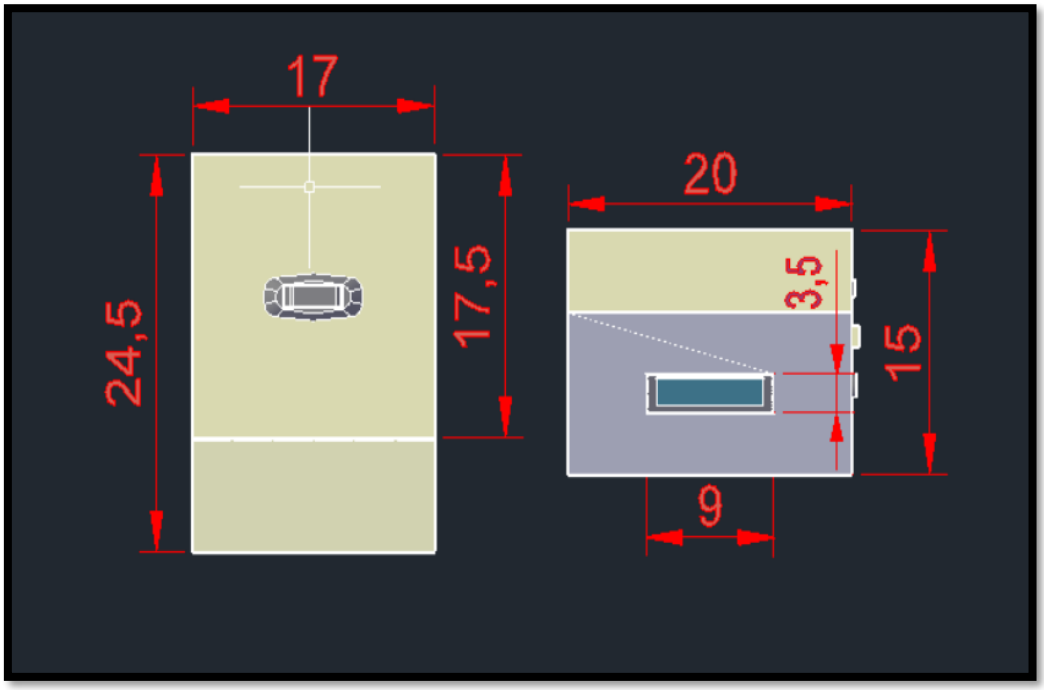
VISTA ISOMETRICA



VISTA INFERIOR



VISTA SUPERIOR



VISTA LATERAL