

# DISEÑO MECATRÓNICO DE UN ROBOT MÓVIL DE DETECCIÓN DE HONGOS Y PLAGAS EN CULTIVOS DE NARANJOS

## MECHATRONIC DESIGN OF A MOBILE ROBOT DETECTING FUNGI AND PESTS IN ORANGE CROPS

Bryan B. Arquíñego Ramirez<sup>1</sup>, Sergio M. Bustinza Gutiérrez<sup>2</sup>, Guiovani A. Gastañaga Bayarri<sup>3</sup>

Ing. Javier Rivas León<sup>4</sup>

### RESUMEN

La población de Lima Metropolitana crece constantemente, al igual que el volumen de personas que migran a esta parte del país, razón por la cual el INEI registró más de 9.700.868 habitantes al 2021, lo que representa el 29,6% de la población total del Perú. Esto provoca que, en la capital, se requiera una gran demanda de alimentos importados de provincia lo que conlleva incapacidad en el control de calidad y deterioro de productos. Por ello, se registran grandes pérdidas y desconfianza en los consumidores. Esta investigación, realizada bajo la supervisión de la escuela de Ingeniería Mecatrónica de la Universidad Ricardo Palma, pretende diseñar un robot móvil detector de hongos y plagas en cultivos que seleccione frutas en mal estado y que las aisle, de tal manera que no contamine parcial o totalmente la producción. El robot móvil consta de un brazo de tres grados de libertad sumamente ligero y de sensores especializados para el reconocimiento biológico y aproximación, que, a través de sus tres orientaciones posibles mediante su gripper, especialmente diseñado para esta función, sujetará eficientemente el fruto a la velocidad requerida en el trabajo agrícola. En el diseño conceptual mecatrónico, se utilizó SolidWorks para el desarrollo de los sistemas mecánicos 3D. En conclusión, se busca que los vendedores brinden frutas de alta calidad a nivel internacional para, de esa manera, obtener certificados de reconocimientos como la ISO 22000. Tras realizarse las pruebas de diseño y simulación, se han obtenido resultados favorables.

**Palabras Claves:** Robot móvil, sensor, detección, detección de plagas, procesamiento de imágenes, radiocontrol, inteligencia artificial

### ABSTRACT

The population of Metropolitan Lima is constantly growing, as is the volume of people who migrate to this part of the country, which is why the INEI registered more than 9,700,868 inhabitants as of 2021, which represents 29.6% of the population. all of Peru. This means that in the capital there is a great demand for food imported from the province, causing incapacity in quality control and deterioration of fruits. Therefore, there are large losses and distrust in consumers. This research, carried out under the supervision of the School of Mechatronics Engineering of the Ricardo Palma University, aims to design a mobile robot that detects fungi and pests in crops that selects fruits in poor condition and isolates them, in such a way that it does not partially or fully production. The mobile robot consists of an arm with three degrees of freedom, extremely light, together with specialized sensors for biological recognition and approach, which, through its three orientations carried out by means of its gripper, specially designed for this function, will efficiently hold the fruit to a required speed of the time necessary to work in said farm. It has a mechatronic conceptual design using SolidWorks for the development of 3D mechanical systems. In conclusion, it is sought that sellers provide high quality fruits and be recognized internationally, in this way, obtain recognition certificates such as ISO 22000 for their great work and seek to bet on said technology in this country's industry. After carrying out the design and simulation tests, favorable results have been obtained.

**Keywords:** Mobile robot, sensor, detection, pest detection, image processing, radio control, artificial intelligence

---

<sup>1</sup> Estudiante del décimo ciclo de Ingeniería Mecatrónica de la Universidad Ricardo Palma. Conocimientos en robótica, diseño gráfico, CAD e inteligencia artificial. <brynarq@gmail.com>

<sup>2</sup> Estudiante del octavo ciclo de Ingeniería Mecatrónica de la Universidad Ricardo Palma. Egresado de la carrera técnica profesional Electrónica y Automatización Industrial de Tecsup. <sergio.bustinza@urp.edu.pe>

<sup>3</sup> Estudiante del octavo ciclo de Ingeniería Mecatrónica de la Universidad Ricardo Palma. Conocimientos en robótica, diseño gráfico y CAD. <guiovani.gastanaga@urp.edu.pe>

<sup>4</sup> Docente de las asignaturas Mecatrónica aplicada al sector agrario, Circuitos Electrónicos e Inmótica de la escuela profesional de Ingeniería Mecatrónica de la Universidad Ricardo Palma. Cuenta con estudios de maestría en Telecomunicaciones, Docencia y Gestión Universitaria; experto en electrónica y geomática. <javier.rivas@urp.edu.pe>

## 1. INTRODUCCIÓN

En la investigación se estableció como estudio de campo la chacra de Pampilla ubicada en San Vicente de Cañete al sur de Lima. En dicho lugar, se tienen áreas destinadas al cultivo de naranjas, por lo que se diseñó un detector con el sistema del robot y una cámara de visión artificial para reconocer las plagas y hongos que pueden afectar a este tipo de plantaciones. Se hicieron trayectos de recorrido de ida y vuelta por cada línea del cultivo.



**Figura 1.** Área de cultivo y recorrido del robot. Fuente: GoogleEarth

En la programación se tuvo en cuenta que el recorrido para el área de cultivo de las naranjas era de aproximadamente de 2.34 km como se muestra en la figura 1.

Este tipo de robots vienen teniendo cada vez más relevancia en el ámbito de la agricultura dada su capacidad para automatizar los procesos agrícolas y efectuar tareas sumamente complejas o repetitivas para un humano [1]. Son capaces de desplazarse por tierra o volar a lo largo del área de cultivo, ya sea a través de un control programado, un control manual dirigido remotamente o por inteligencia artificial. En general, los robots agrícolas se valen de una serie de dispositivos y tecnologías para su adecuado funcionamiento como sensores de obstáculos, GPS, módulos de radiocontrol, sensores infrarrojos, conexión WiFi, brazos de múltiples grados de libertad, cámaras, etc.

El diseño propuesto consiste en un robot móvil dotado con un brazo de 3 grados de libertad que cuenta con una cámara en su extremo, la cual permite la adquisición de imágenes para su posterior análisis a través de algoritmos de visión artificial.

El robot es radiocontrolado desde una estación en tierra, la cual recibe las imágenes y las alertas detectadas por el robot, su cámara y sus sensores. De igual manera que con el control de las ruedas tipo *mecanum* para el desplazamiento, el brazo puede ser radiocontrolado en cada uno de sus grados articulares.

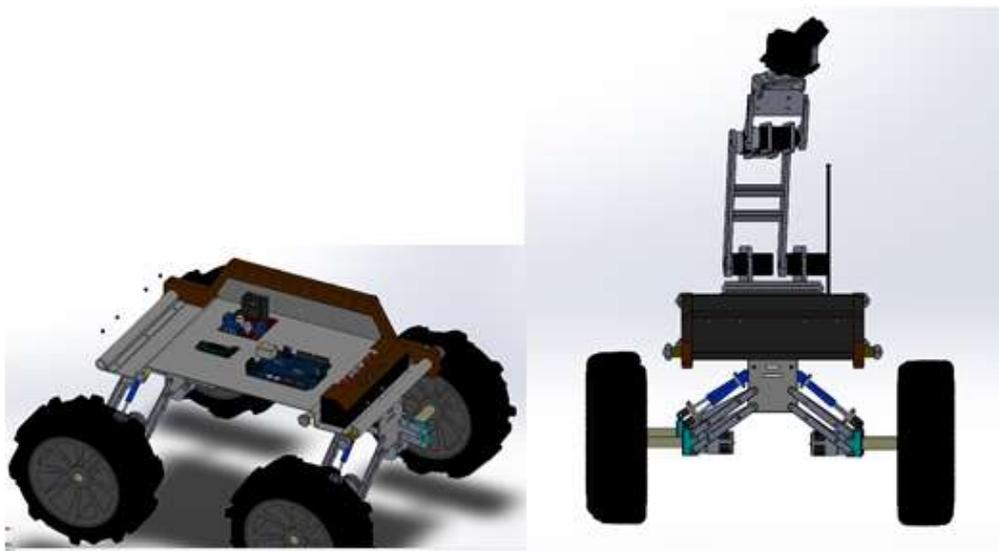


**Figura 2.** Muestra del robot Elaborado en SolidWork

## 2. MATERIALES Y MÉTODOS

### 2.1. Diseño mecánico

Internamente cuenta con los circuitos del controlador, los motores los sensores ultrasónicos y la batería [3]. Mediante la vista frontal se observan los suspensores [2] para evitar que se caiga y mantenga un equilibrio en cualquier tipo de zona que pueda recorrer, tal como se muestra en las vistas de la figura 3.



**Figura 3.** Vista interna y frontal del robot. Elaborado en SolidWork

En la figura 4, se muestra el brazo robótico de 3 GDL con una cámara [4].

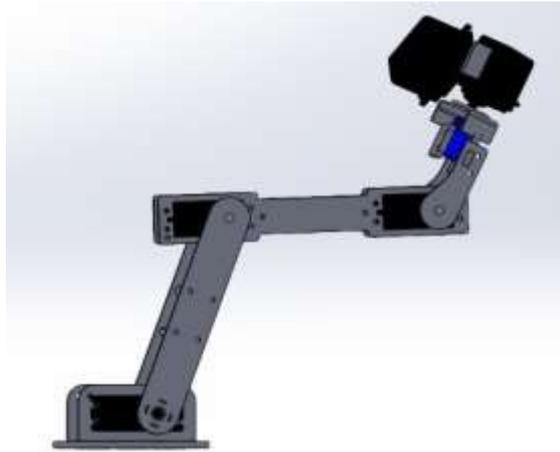


Figura 4. Vista lateral del robot. Elaborado en SolidWork

## 2.2. Diseño eléctrico y electrónico

El robot cuenta con diversos subcircuitos que se comunican a un microcontrolador central (un ATMEGA 328P) que gobierna las diferentes funciones del sistema [10]. Dicho sistema posee un circuito para las ruedas, un circuito para los servomotores del brazo, un circuito para los sensores, uno para la cámara y otro para la comunicación por radiocontrol.

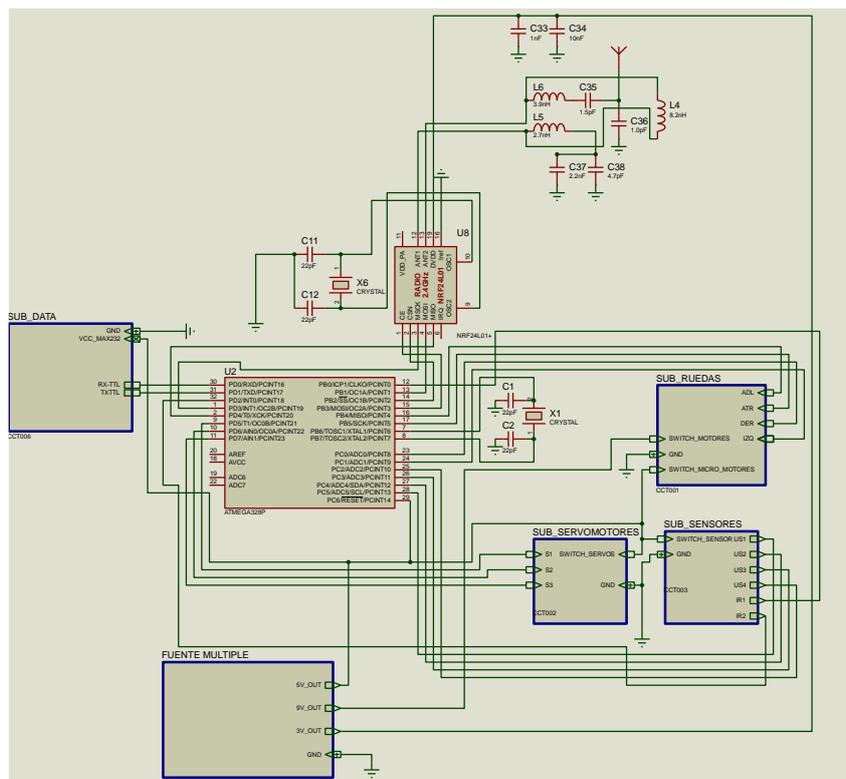
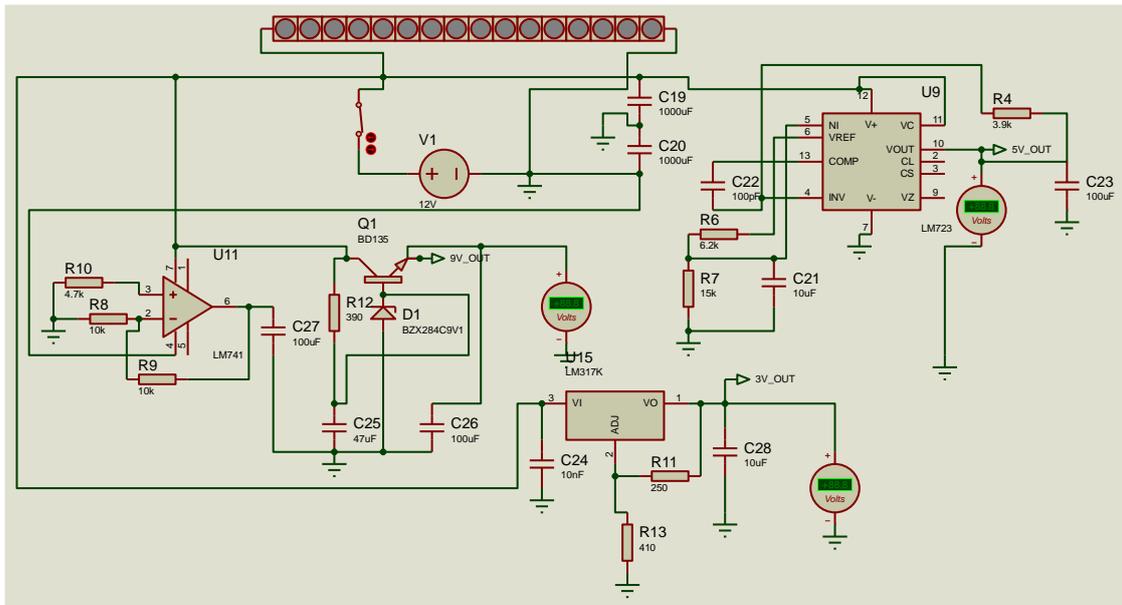


Figura 5. Esquema eléctrico-electrónico del robot móvil. Elaborado en Proteus

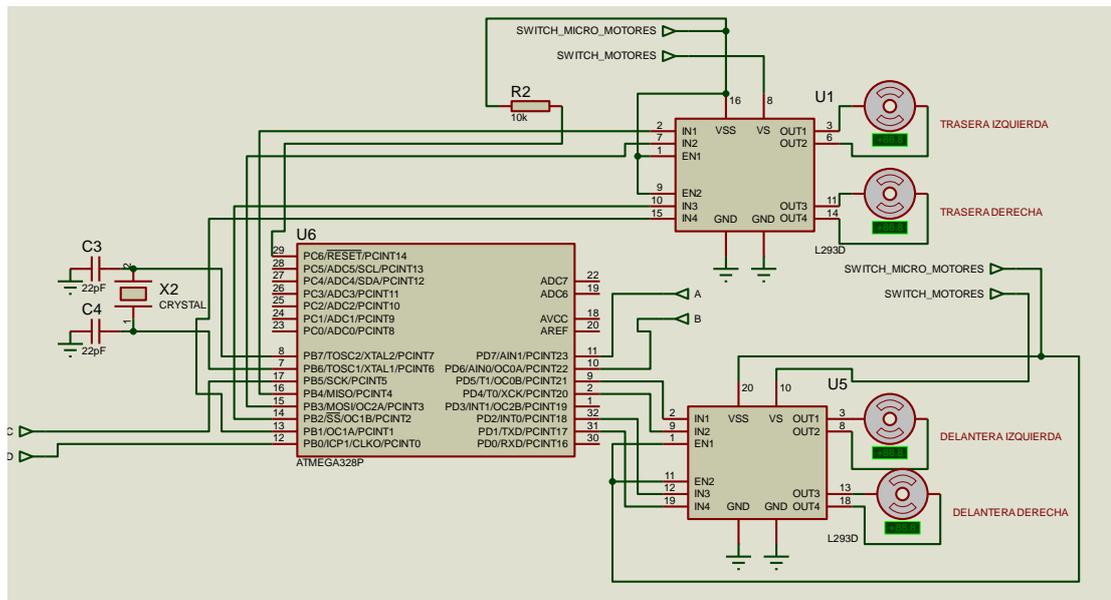
La batería de 12V 5AH suministra energía a un circuito de alimentación múltiple, entrega diferentes valores de tensiones adaptables a las diferentes necesidades de los módulos. La

fuentes de alimentación múltiple hace uso de un LM723, un LM 741 y un LM317K para entregar 5V, 9V y 3.3V, respectivamente [13]. Asimismo, cuenta con unos indicadores leds que se activan al encenderse el circuito de alimentación por medio del switch.



**Figura 6.** Esquema de la fuente de alimentación múltiple. Elaborado en Proteus

El sistema de ruedas se basa en el uso de un microcontrolador ATMEGA 328P, dos integrados puente H L293D para el control de la dirección de los motores y 4 motores de 9V.



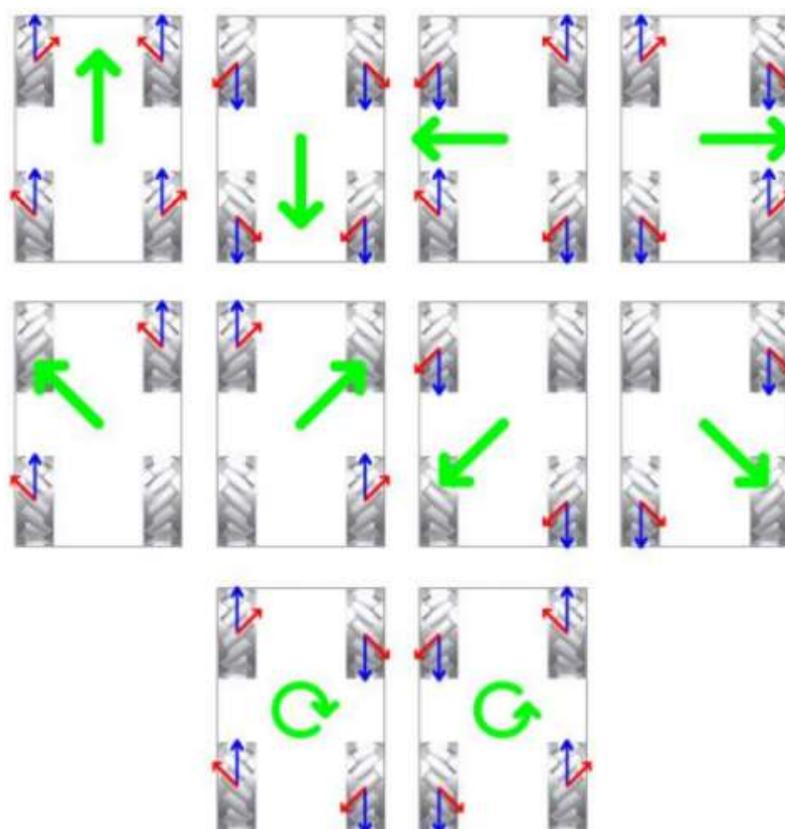
**Figura 7.** Circuito de accionamiento de las ruedas. Elaborado en Proteus

Al circuito ingresan cuatro entradas del microcontrolador principal del sistema (A, B, C,

D), que corresponden a las señales ordenadas vía radiocontrol desde la estación en tierra, que indicarán de manera codificada cómo deben girar los motores y, en consecuencia, determinarán el desplazamiento del robot. La codificación se expone a continuación:

A	B	C	D	MOVIMIENTO
0	0	0	0	NO SE MUEVE
0	0	0	1	ADELANTE
0	0	1	0	ATRÁS
0	0	1	1	DERECHA
0	1	0	0	IZQUIERDA
0	1	0	1	ADELANTE-IZQUIERDA
0	1	1	0	ADELANTE-DERECHA
0	1	1	1	ATRÁS-IZQUIERDA
1	0	0	0	ATRÁS-DERECHA
1	0	0	1	GIRO HORARIO
1	0	1	0	GIRO ANTIHORARIO

**Tabla 1.** Codificación de los movimientos del robot



**Figura 8.** Movimiento de las ruedas mecanum [9]. Fuente: IES Pedro de Valdivia

El subcircuito de servomotores funciona de manera homóloga al subcircuito de las ruedas. Cuenta con un microcontrolador ATMEGA 328P, tres servomotores de 5V (uno para cada articulación del brazo robótico) y tres entradas de control provenientes del microcontrolador principal. Las entradas determinan de manera codificada cómo se combinan los movimientos de los servomotores como se precisa a continuación:

S1	S2	S3	H1	H2	H3	A1	A2	A3
0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0	0
0	1	0	0	1	0	0	0	0
0	1	1	0	0	0	1	0	0
1	0	0	1	0	0	0	0	0
1	0	1	0	0	0	0	1	0
1	1	0	0	0	0	0	0	1
1	1	1	0	0	0	0	0	0

**Tabla 2.** Codificación de los sentidos de giro de los servomotores

S1, S2 y S3 son las entradas de control correspondientes a las señales provistas de radiocontrol para la dirección del brazo. H1, H2 y H3 corresponden a los giros horarios del servomotor 1, del servomotor 2 y del servomotor 3, respectivamente. A1, A2 y A3 corresponden a los giros antihorarios del servomotor 1, del servomotor 2 y del servomotor 3, respectivamente.

El circuito de los sensores se caracteriza por la implementación de 2 sensores infrarrojos y 4 ultrasónicos. Los sensores infrarrojos van acoplados en la parte frontal del robot y en el extremo del brazo robótico y permiten la detección de plantas, mientras que los sensores ultrasónicos, dispuestos cada uno a cada lado del robot, hacen posible la detección de obstáculos. Los sensores infrarrojos operan sobre un valor de 0.8  $\mu\text{m}$  de longitud de onda (longitud de onda de la radiación emitida por una planta [5]); y los sensores ultrasónicos, sobre los 20  $\mu\text{m}$ .

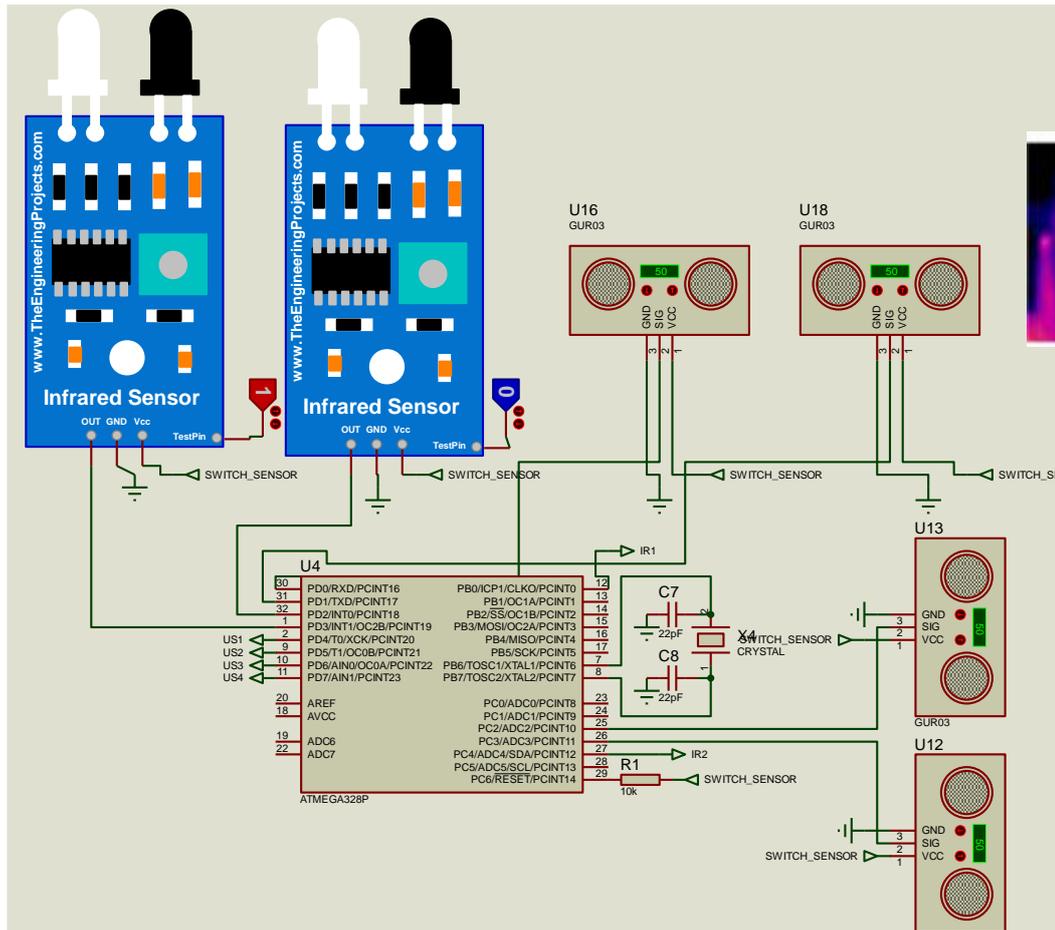


Figura 9. Circuito de sensores Elaborado en Proteus

El circuito se comunica por protocolo UART con el microcontrolador principal y se conecta a la cámara por un módulo que compatibiliza el protocolo TTL, con el que trabajan los microcontroladores, con el protocolo RS-232, con el que opera la cámara. Dicho módulo es el MAX232 [15].

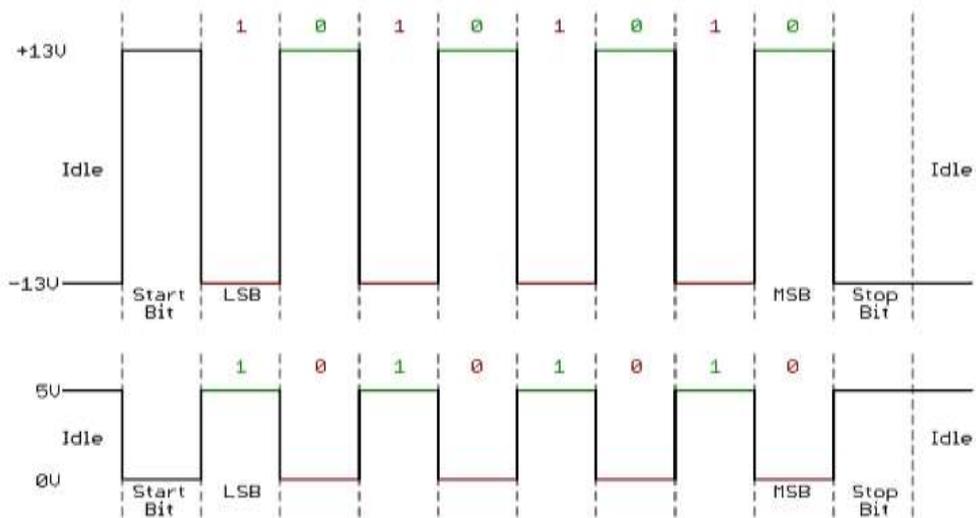
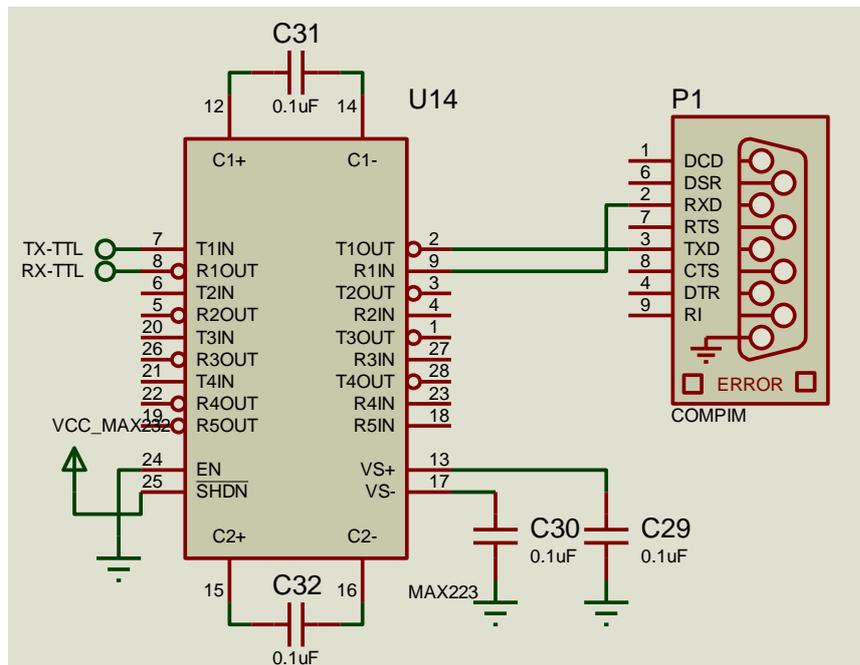
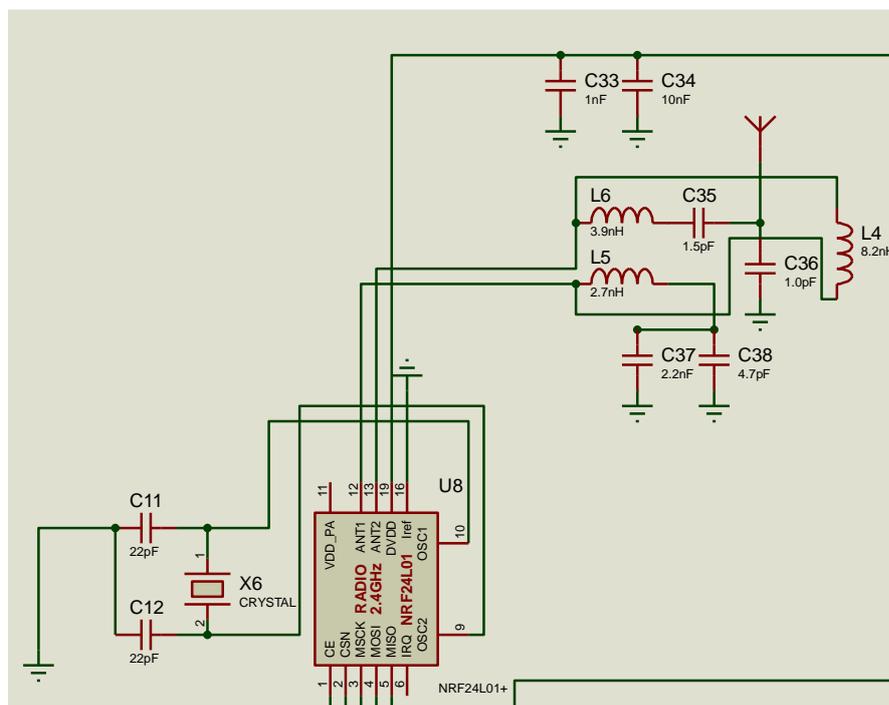


Figura 10. TTL vs RS-232 [16]



**Figura 11.** Circuito de comunicación entre la cámara y el microcontrolador principal.

Para la comunicación por radiocontrol, se cuenta con dos módulos de radiofrecuencia NRF24L01+ (capaz de transmitir y recibir información). Uno está instalado en el robot y otro en la estación en tierra [11]. El módulo instalado en el robot se encarga de recibir las órdenes de control de las ruedas y del brazo robótico, a la vez que transmite la información de los sensores y la cámara. Por otro lado, el módulo en tierra emite las órdenes de control móvil y recibe las alertas de los sensores y la información de la cámara.



**Figura 12.** Módulo de radiocontrol conectado al microcontrolador principal. Elaborado en Proteus

La estación tierra cuenta con el módulo de radiocontrol, dos mandos para la dirección a distancia del robot y el brazo robótico, que funcionan bajo los sistemas de codificación ya propuestos, una matriz de leds 8X8 que indica el movimiento que describe el robot móvil y está conectado a un driver para display MAX7219, una serie de leds que corresponden a las alertas de los 6 sensores implementados en el robot y un microcontrolador con comunicación USB ATMEGA 32U4, que envía la información de la cámara a una computadora vía USB [12].

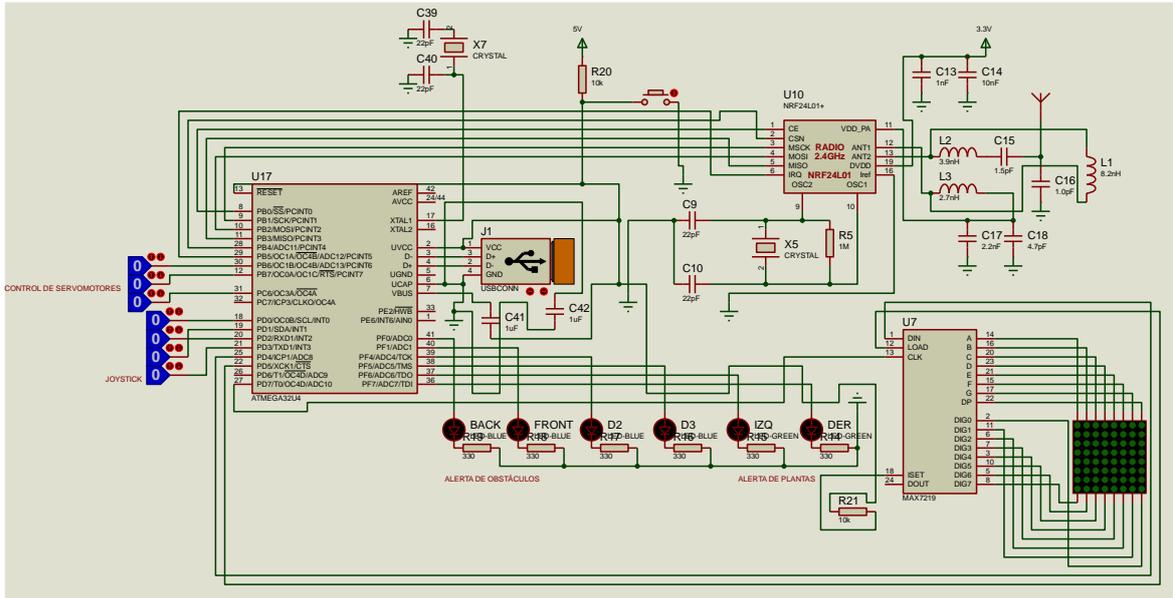


Figura 13. Circuito de estación en tierra. Elaborado en Proteus

### 2.3. Procesamiento de imágenes por inteligencia artificial YOLO

Para la detección del procesado de imagen, toda la programación se realizó mediante GOOGLE COLABORATORY y JUPITER [7]. Para ello, se almacenaron las librerías en la nube del GOOGLE DRIVE. Luego se procedió a descargar las imágenes que se buscan entrenar para su detección; en este caso, se descargaron imágenes de frutas en mal estado como las naranjas con *Penicillium italicum Wehme* y se guardaron en una carpeta llamada “all images”, la cual interactúa con las librerías de JUPITER programadas como “rename\_files”, para renombrar los nombres de las imágenes descargadas y ordenarlas de manera numérica creciente, y “split\_file”, para separar las imágenes para el reconocimiento del entrenamiento (train), la validación (validation) y la prueba (test).

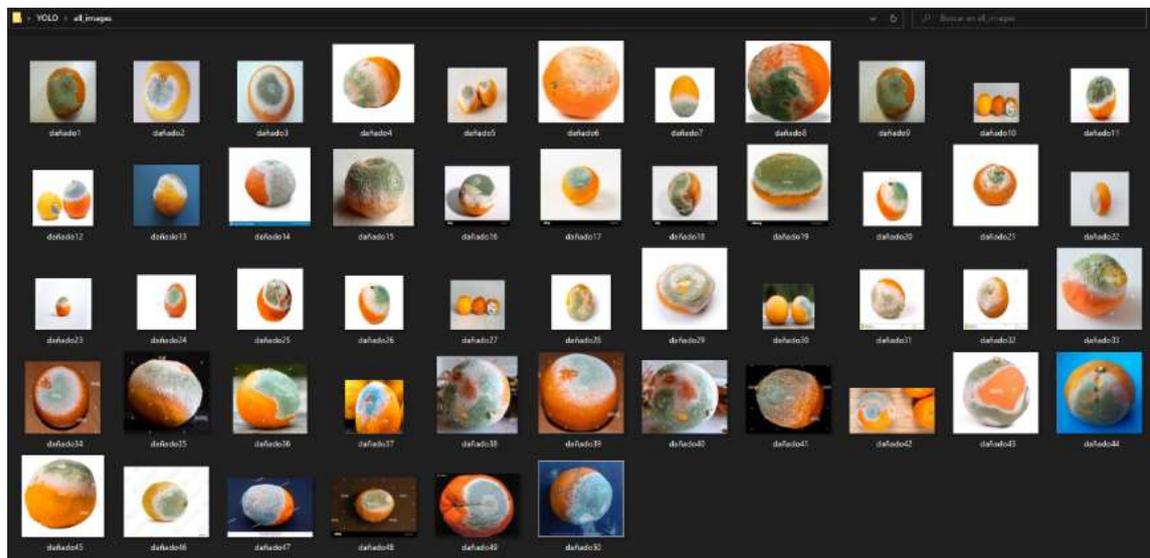


Figura 14. Imágenes de naranjas con *Penicillium italicum* Wehme

En la figura 15, se muestra el entorno de programación de la aplicación Jupyter Notebook.

#### Script para cambiar el nombre a las imágenes que se encuentran en una carpeta o directorio

```

In [4]: #Importamos las librerías para el manejo de archivos
import os
import random
from shutil import copyfile
import shutil

In [5]: #Colocar el nombre del directorio
dir = "all_images"
#Colocar el nombre de nuestra clase (No usar espacios)
name_class="hongos"

Correr solo una vez

In [8]: count = 0
list_files=os.listdir(dir)
new_dir=dir+"_rename"

if not os.path.exists(new_dir):
    os.makedirs(new_dir)
else :
    shutil.rmtree(new_dir)

for filename in list_files:
    if filename.endswith(".jpg"):
        new_name=name_class+"_"+str(count)+"_"+filename.split(".")[1]
        copyfile(os.path.join(dir, filename),os.path.join(new_dir, new_name ))
        print(filename,"->", new_name)
        count += 1
print("Archivos JPG totales", count)

Archivos JPG totales 54

```

#### Script para separar las imágenes

```

In [6]: #Importamos la librería para el manejo de archivos
import os
import random
from shutil import copyfile
import shutil

In [7]: def img_train_test_split(img_source_dir, train_size, validation_size):
    """
    Parametros
    img_source_dir : string
    Estructura de las imágenes

    train_size : float
    Porcentaje de la muestra de entrenamiento, ejemplo 0.80 (80%)

    validation_size : float
    Porcentaje de la muestra de validación, ejemplo 0.15 (15%)

    El restante 5% quedaria para el numero de imágenes de prueba, imágenes que nunca ha visto el modelo

    # Configurar la estructura de carpetas vacías si no existe
    if not os.path.exists('dataset'):
        os.makedirs('dataset')

    # Crear subestructuras en carpetas de entrenamiento, validación y test
    if not os.path.exists('train_subdir'):
        os.makedirs('train_subdir')

    # Contar el número de imágenes totales
    count_images=0
    for filename in os.listdir(subdir_fullpath):
        if filename.endswith(".jpg"):
            #Número de muestra aleatoria (en imágenes)
            list_files=os.listdir(subdir_fullpath)
            random.shuffle(list_files)
            #Separación de imágenes
            for filename in list_files:
                if filename.endswith(".jpg"):
                    #Copiar el archivo classes.txt a las diferentes directorios pero con otro nombre
                    copyfile(os.path.join(subdir_fullpath, 'classes.txt'), os.path.join(train_subdir, "dataset_label1"))
                    copyfile(os.path.join(subdir_fullpath, 'classes.txt'), os.path.join(validation_subdir, "dataset_label1"))
                    copyfile(os.path.join(subdir_fullpath, 'classes.txt'), os.path.join(test_subdir, "dataset_label1"))

In [17]: # 1: Directorio de las imágenes
# 2: Porcentaje de la muestra de entrenamiento, ejemplo 0.80 (80%)
# 3: Porcentaje de la muestra de validación, ejemplo 0.15 (15%)
# 4: Rescorta el tamaño de las imágenes de prueba

img_train_test_split("all_images_rename", 0.80,0.15)

48
Copied 37 images to dataset/train/
Copied 7 images to dataset/validation/
Copied 2 images to dataset/test/

```

Figura 15. *Rename\_files*, y *split\_file*

Posteriormente se utiliza el software “LabelImg”, se selecciona la carpeta “all\_images”, en modo de algoritmo YOLO, y luego se seleccionó el área de detección mediante “CreateRectBox”. Asimismo, se etiquetaron las imágenes con el nombre de “danado” para todos los jpg’s y se guardaron en “save” de manera automática.

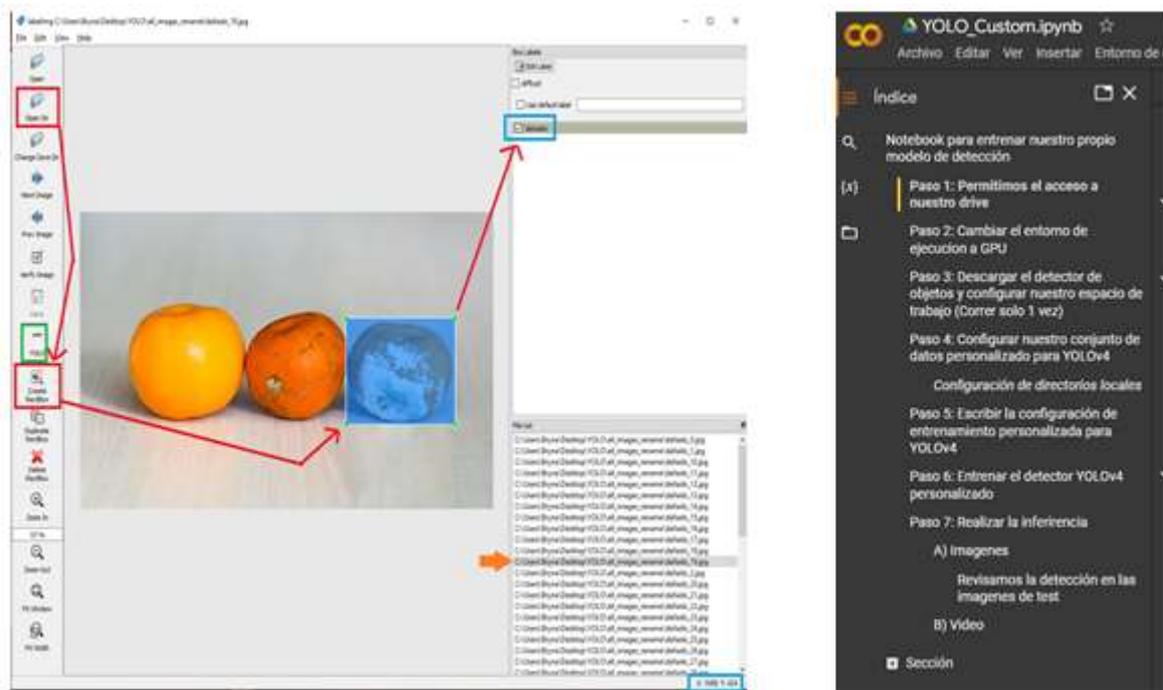


Figura 16. Procedimiento de uso del software Labellmg e Índice de Yolo\_Custom.

Una vez terminado este procedimiento, se puede entrenar en Google Colaboraty al algoritmo YOLO, para lo cual se desarrolló un índice del orden de los pasos correspondientes en su compilación. Los primeros pasos sirvieron para obtener el permiso de usar las carpetas de imágenes cargadas al Drive, todo en GPU para que administre la memoria, ya que de esa manera el usuario consigue un mayor rendimiento y una menor latencia. Posteriormente, se descargó el detector, instalando el “**darknet**”, el cual cumple un papel muy importante para la detección de las frutas.

```

1 from ctypes import *
2 import math
3 import random
4 import os
5 import cv2
6 import numpy as np
7 import time
8 import darknet

frame_rgb = cv2.cvtColor(frame_read, cv2.COLOR_BGR2RGB)
frame_resized = cv2.resize(frame_rgb,
                           (darknet.network_width(netMain),
                            darknet.network_height(netMain)),
                           interpolation=cv2.INTER_LINEAR)

#cap = cv2.VideoCapture(0)
cap = cv2.VideoCapture("test.mp4")
cap.set(3, 1280)
cap.set(4, 720)
out = cv2.VideoWriter(
    "output.avi", cv2.VideoWriter_fourcc("MJPG"), 10.0,

```

Figura 17. Programación del Darknet

En esta etapa, se procedió a descargar el código del **Darknet** de manera directa al entorno de programación y se configuraron sus datos personalizados de entrenamiento para YOLO como lo son la carpeta de “test”, “train” y “valid”.

```

Paso 4: Configurar nuestro conjunto de datos personalizado para YOLOv4

Usaremos las imágenes que previamente han sido recolectadas y colocadas en los directorios train, valid y test en el archivo llamado
imagenes.zip.

[6] #Nos aseguramos de entrar al directorio de trabajo darknet
cd /content/darknet

/content/darknet

[7] #Verificamos tener nuestros archivos
ls /content/drive/My_Drive/YOLO/custom/data

danadi_46_detectado.jpg  naranjas0.mp4  test  web_scraping.ipynb
danadi_9_detectado.jpg  naranjas1.mp4  train
imagenes.zip             naranjas2.mp4  valid

[8] # Copiamos las carpetas test, train y valid a nuestro espacio de trabajo
cp -r /content/drive/My_Drive/YOLO/custom/data/test /content/darknet
cp -r /content/drive/My_Drive/YOLO/custom/data/train /content/darknet
cp -r /content/drive/My_Drive/YOLO/custom/data/valid /content/darknet

```

Figura 18. Darknet reconociendo las carpetas de las imágenes de las frutas

Se realiza la compilación para el Entrenamiento del detector YOLO personalizado. Se puede destacar el ahorro del tiempo estimado para completar la detección, así como su porcentaje de completado en el reconocimiento. En este caso, llegó a completarse hasta un 85.71% de los datos adquiridos.

```

Paso 6: Entrenar el detector YOLOv4 personalizado

[13] !./darknet detector train data/obj.data cfg/custom-yolov4-detector.cfg yolov4.conv.137 -dont_show -map

(next mAP calculation at 2000 iterations)
Last accuracy mAP@0.5 = 85.71 %, best = 85.71 %
1994: 0.134825, 0.179466 avg loss, 0.000010 rate, 0.548449 seconds, 95712 images, 0.182132 hours left
loaded: 0.000041 seconds

(next mAP calculation at 2000 iterations)
Last accuracy mAP@0.5 = 85.71 %, best = 85.71 %
1995: 0.104835, 0.179103 avg loss, 0.000010 rate, 0.550248 seconds, 95760 images, 0.180453 hours left
loaded: 0.000034 seconds

(next mAP calculation at 2000 iterations)
Last accuracy mAP@0.5 = 85.71 %, best = 85.71 %
1996: 0.109586, 0.180151 avg loss, 0.000010 rate, 0.647664 seconds, 95808 images, 0.178767 hours left
loaded: 0.000047 seconds

(next mAP calculation at 2000 iterations)
Last accuracy mAP@0.5 = 85.71 %, best = 85.71 %
1997: 0.227825, 0.184839 avg loss, 0.000010 rate, 0.549005 seconds, 95856 images, 0.177076 hours left
loaded: 0.000036 seconds

(next mAP calculation at 2000 iterations)
Last accuracy mAP@0.5 = 85.71 %, best = 85.71 %
1998: 0.141266, 0.180482 avg loss, 0.000010 rate, 0.422838 seconds, 95904 images, 0.175376 hours left
loaded: 0.000044 seconds

(next mAP calculation at 2000 iterations)
Last accuracy mAP@0.5 = 85.71 %, best = 85.71 %
1999: 0.113270, 0.173760 avg loss, 0.000010 rate, 0.418132 seconds, 95952 images, 0.173669 hours left
loaded: 0.000040 seconds

(next mAP calculation at 2000 iterations)
Last accuracy mAP@0.5 = 85.71 %, best = 85.71 %
2000: 0.120671, 0.168451 avg loss, 0.000010 rate, 0.315757 seconds, 96000 images, 0.171956 hours left
Resizing to initial size: 416 x 416 try to allocate additional workspace_size = 52.43 MB
CUDA allocate done!

```

Figura 19. Programación para el entrenamiento del detector YOLO

Luego, para el reconocimiento de imágenes entrenadas en YOLO, se inserta “imshow” para mostrar la imagen en una ventana, también se importa “cv2” para la introducción del OpenCV, y se importa también “matplotlib.pyplot” para crear el gráfico en dos dimensiones, X y Y, que tiene el conjunto de imágenes.

```

- Paso 7: Realizar la inferencia

- A) Imagenes

[17] #Definimos la función que nos desplegará las imágenes
def imshow(path):
    import cv2
    import matplotlib.pyplot as plt
    %matplotlib inline

    image = cv2.imread(path)
    height, width = image.shape[:2]
    resized_image = cv2.resize(image,(3*width, 3*height), interpolation = cv2.INTER_CUBIC)

    fig = plt.gcf()
    fig.set_size_inches(18, 18)
    plt.axis("off")
    %plt.rcParams['figure.figsize'] = [18, 5]
    plt.imshow(cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB))
    plt.show()

```

Figura 20. Programación para el reconocimiento de imágenes

Luego, para el reconocimiento de los videos entrenados en YOLO, se agrega “!cp” para el copiado de los ficheros y directorios, como también “!ls” para que enliste los archivos actuales. En este caso, el vídeo es “naranja0\_detectado” y “fresa0\_detectado” para su detección en tiempo real.

```

- B) Video

[18] #Copiar en otro para probar
!cp "/content/drive/MyDrive/YOLO4/custom_data/naranja0" "/content/naranja0"

[19] #Verificamos que tengamos el video
!ls *.mp4

fresa0_detectado.mp4  naranja0.mp4  naranja0_detectado.mp4
fresa0.mp4           naranja0_detectado.mp4  naranja0.mp4
fresa0_detectado.mp4  naranja0.mp4  naranja0_detectado.mp4
fresa0.mp4           naranja0_detectado.mp4  naranja0.mp4
naranja0_detectado.mp4  naranja0.mp4  naranja0.mp4

#Ejecutar en video
!./darknet_detector.exe data/obj.data cfg/custom-yolov4-detector.cfg backup/custom-yolov4-detector_best.weights naranja0.mp4 -thresh 0.3 -dont_show -out_filename naranja0_detectado.mp4

```

Figura 21. Programación para el reconocimiento de vídeos

### 3. PRUEBAS Y RESULTADOS

#### Prueba 1

Como se observa en la figura 23, los resultados fueron óptimos gracias al entrenamiento del detector, ya que se logró identificar dentro del cuadro la sección dañada de la fruta.

```

# Utilizamos las imágenes de la carpeta /test para probar nuestro modelo
test_images = [f for f in os.listdir('test') if f.endswith('.jpg')]
image=test_images[1]
img_path = 'test/' + image;
#test out our detector!
!./darknet detect cfg/custom-yolov4-detector.cfg backup/custom-yolov4-detector_best.weights {img_path} -thresh 0.3 -dont_show
imshow('predictions.jpg')
print("*****")
print("*****",image,"*****")
print("*****")

```

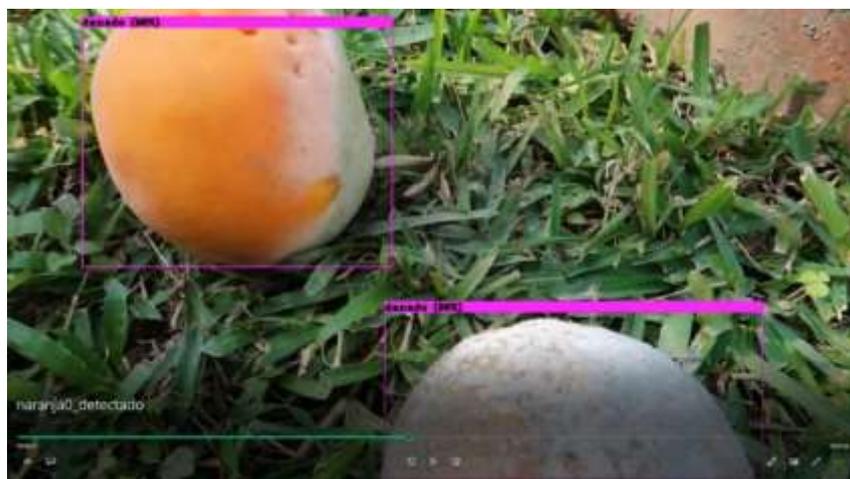
Figura 22. Resultados de la simulación



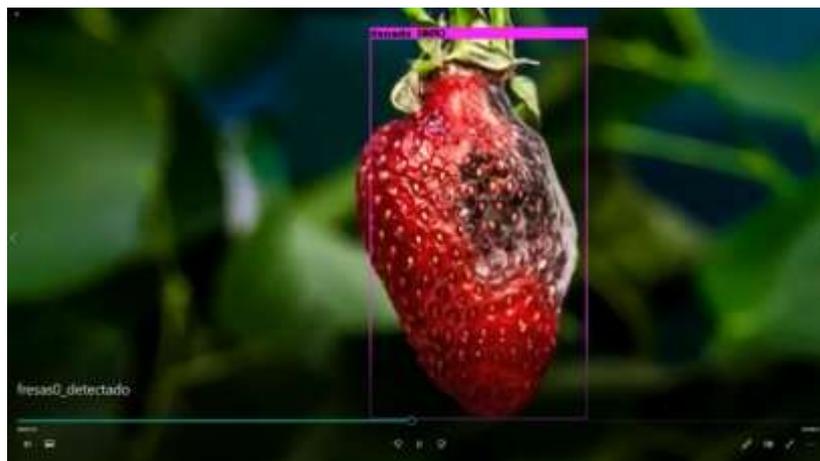
**Figura 22.** Imagen de detección de una naranja con *Penicillium italicum* Wehme

## **Prueba 2**

En cuanto al resultado del vídeo subido al drive para su respectiva detección, los resultados también fueron óptimos en su detección de las naranjas y fresas con *Penicillium italicum* Wehme. Esto se debe a que el sensor Kinect realiza la detección de las frutas en tiempo real.



**Figura 23.** Reconocimiento de una naranja con *Penicillium italicum* Wehme



**Figura 24.** Reconocimiento de una fresa con *Penicillium italicum* Wehme

## 4. CONCLUSIONES

El robot móvil diseñado permite la detección de hongos y plagas en cultivos de naranjos mediante un desplazamiento telecontrolado capaz de tomar imágenes que fueron procesadas mediante inteligencia artificial YOLO.

Los procesos de entrenamiento al algoritmo YOLO para el reconocimiento de imágenes y secciones de cuadros en video permiten resultados efectivos, puesto que permiten detectar anomalías en las plantaciones de acuerdo con la clasificación de etiquetas, estados y alarmas detectados en las secuencias de imágenes captadas por el robot.

El robot móvil es de suma utilidad para la industria agrícola porque permite la automatización de las tareas programadas y la inspección más precisa del estado y condiciones de los cultivos. De esta manera, este tipo de tecnologías tiene injerencia directa en la calidad y eficacia del monitoreo de los productos agrícolas.

## 5. REFERENCIAS

- [1] EDS ROBOTICS (s.f.). Agricultura automatizada y robótica agrícola. Recuperado 08 de julio de 2022 de <https://www.edsrobotics.com/blog/agricultura-automatizada-y-robotica-agricola/#:~:text=Un%20robot%20agr%C3%ADcola%20es%20uno,recorridos%20seguros%20de%20los%20campos.>
- [2] Solidworks Fun (2021). RC Car with Suspension System All Parts Design Tutorial in Solidworks. [video] Recuperado el 12 de junio de 2022 de: [https://www.youtube.com/watch?v=uEnT38EBxp4&t=3486s&ab\\_channel=SolidworksFun](https://www.youtube.com/watch?v=uEnT38EBxp4&t=3486s&ab_channel=SolidworksFun)
- [3] Cagatay, O. (2017). Robotic Tank Unmanned Ground Vehicle UGV. GrabCad. Recuperado el 20 de junio del 2022 de: <https://grabcad.com/library/robotic-tank-unmanned-ground-vehicle-ugv-1>
- [4] Sadeghi, H. (2019). Small-Raspberry-Picker-Robot. Grabcad. Recuperado el 20 de junio del 2022 de: <https://grabcad.com/library/small-raspberry-picker-robot-1>
- [5] Universidad de Murcia (s.f.). Interacción de la radiación con los objetos. Recuperado de 08 de julio de 2022 de <https://www.um.es/geograf/sigmur/teledet/tema02.pdf>.
- [6] Camacho, E. (2021, 31 de enero). 2. Descargar y cortar un Video de YouTube (Detección de objetos con YOLO y Google-COLAB)
- [7] Camacho, E. (2021, 31 de enero). Probando el detector YOLO en imágenes y videos (Detección de objetos con YOLO y Google-COLAB:) (Detección de objetos con YOLO y Google-COLAB:) [Video]. Youtube. [https://www.youtube.com/watch?v=DUMdil54rEk&ab\\_channel=EnriqueCamacho](https://www.youtube.com/watch?v=DUMdil54rEk&ab_channel=EnriqueCamacho)
- [8] Camacho, E. (2021, 31 de enero). 4. Personalizando el detector YOLO (Detección de objetos con YOLO y Google-COLAB:) [Video]. Youtube.

[https://www.youtube.com/watch?v=DUMdil54rEk&ab\\_channel=EnriqueCamacho](https://www.youtube.com/watch?v=DUMdil54rEk&ab_channel=EnriqueCamacho)

- [9] IES Valdivia (s.f.). Mecanum Robot. Recuperado el 08 de julio de 2022 de <https://lh5.googleusercontent.com/30IHFQWEbC7sdAAe4rRfD-GjjiV03tnjbVnifWDts48cG6LHnYLWN0iOs9wVbyhgUTA02HDbGGXzL9zjBYiyTZmQa4qn5MWarQEfkgbFImqN2PFbIUFWNzoTqYW0wEBfYCJFK9Vy>
- [10] Arduino.cc (s.f.). ATmega168/328-Arduino Pin Mapping. Recuperado el 28 de junio de 2022 de <https://assiss.github.io/arduino-zhcn/cn/Hacking/PinMapping168>.
- [11] Nordic Semiconductor (2006). Single chip 2.4 GHz Transceiver nRF24L01. Recuperado el 05 de julio de 2022 de <https://pdf1.alldatasheet.com/datasheet-pdf/view/1243924/ETC1/NRF24L01.html>.
- [12] Electrónica para todos (s.f.). Librerías de Sensores para Microcontroladores PIC en PROTEUS. Recuperado el 28 de junio de 2022 de <http://www.electronicworld.com.mx/electronica/librerias-de-sensores-para-microcontroladores-pic-en-proteus/>.
- [13] Braga, Newton (s.f.). Fuente múltiple 3, 5 y 9 V (CIR11122S). Recuperado el 28 de junio de 2022 de <http://www.incb.com.mx/index.php/banco-de-circuitos/17526-fuente-multiple-3-5-y-9-v-cir11122s>.
- [14] Componentes 101 (2021). MAX3232 - RS232 to TTL Serial Port Converter Module. Recuperado el 05 de julio de 2022 de <https://components101.com/modules/max3232-rs232-to-ttl-serial-port-converter-module>.
- [15] Maxim Integrated (2019). MAX3222/MAX3232/MAX3237/MAX3241\*. Recuperado el 05 de julio de 2022 de <https://pdfserv.maximintegrated.com/en/ds/MAX3222-MAX3241.pdf>.
- [16] Sparkfun (2010). RS-232 vs. TTL Serial Communication. Recuperado el 05 de julio de 2022 de <https://www.sparkfun.com/tutorials/215>.