

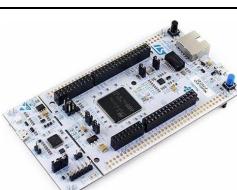
**Project name** Sensor para maduración de frutas con RGB**Autor(S)** T025997, Emiliano Machado  
T025605, Gustavo Salazar**Editor** Prof. Nataly Medina**Last update** 7 de diciembre del 2023

## Problem statement

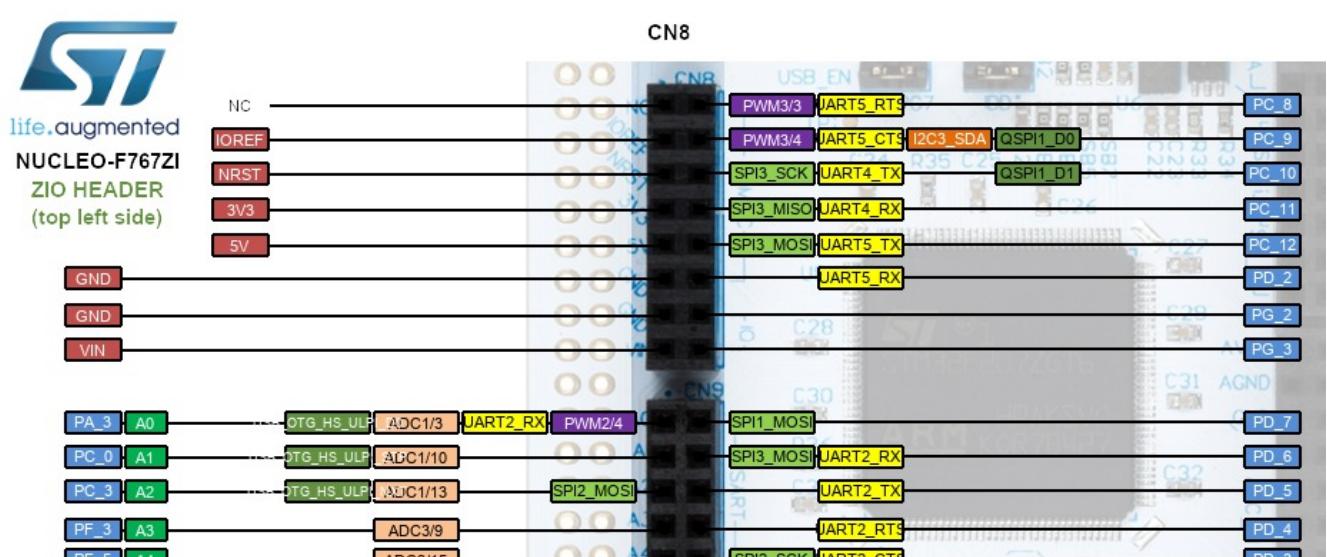
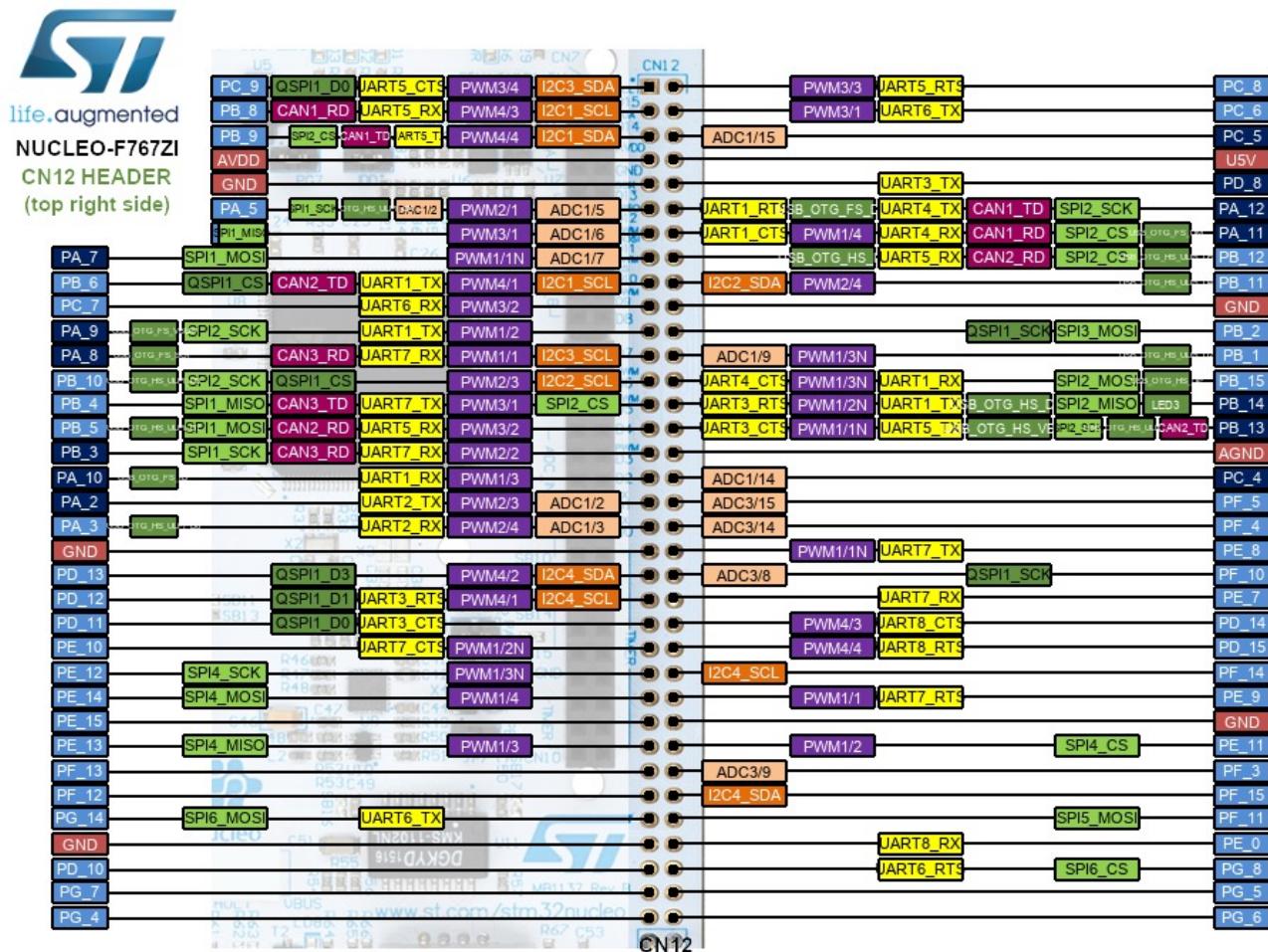
El daltonismo, una condición visual que afecta la percepción de ciertos colores, ha sido objeto de estudio y atención debido a sus implicaciones en diversos aspectos de la vida cotidiana, especialmente en entornos digitales. En este informe, exploramos la relación entre el daltonismo y la representación de colores en sistemas RGB (Red, Green, Blue), comúnmente utilizados en dispositivos electrónicos y multimedia.

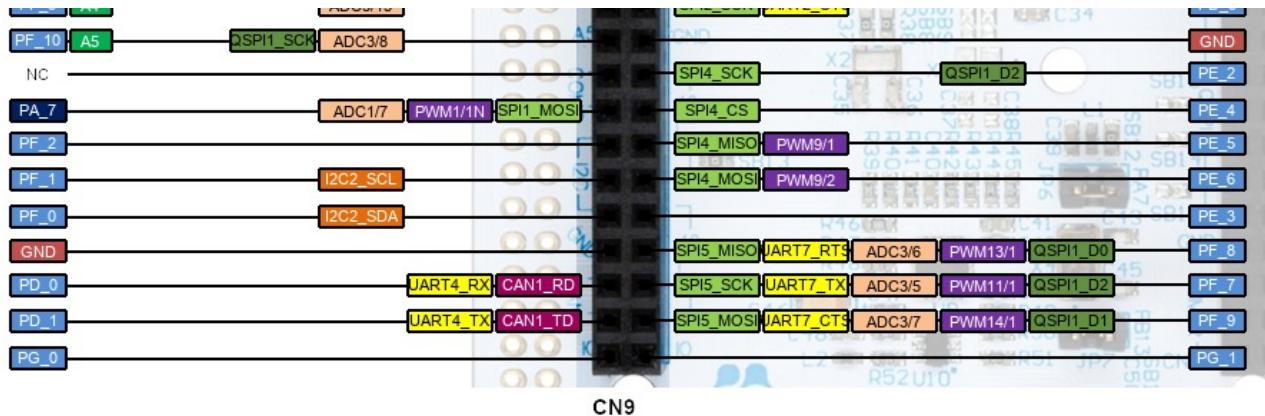
El daltonismo es una anomalía hereditaria o adquirida en la visión del color, afectando la capacidad de distinguir ciertos tonos. Las personas con daltonismo experimentan dificultades para percibir colores específicos, siendo el rojo y el verde los más comúnmente afectados. Dada la prevalencia del modelo de color RGB en pantallas, cámaras y otros dispositivos, es esencial comprender cómo esta elección de representación de colores puede afectar a las personas con daltonismo y qué estrategias se pueden implementar para mejorar su experiencia visual.

## Requisitos de Hardware

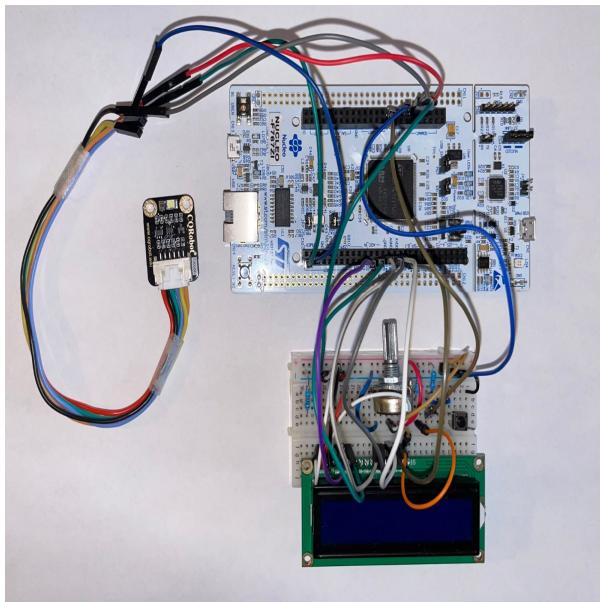
Component	Quantity	Characteristics	Component
STM32 F767ZI	1	Development board	
TCS34725FN	1	Sensor RGB	
LCD 16x2	1	Pantalla LCD	

## ✓ Hardware schematic

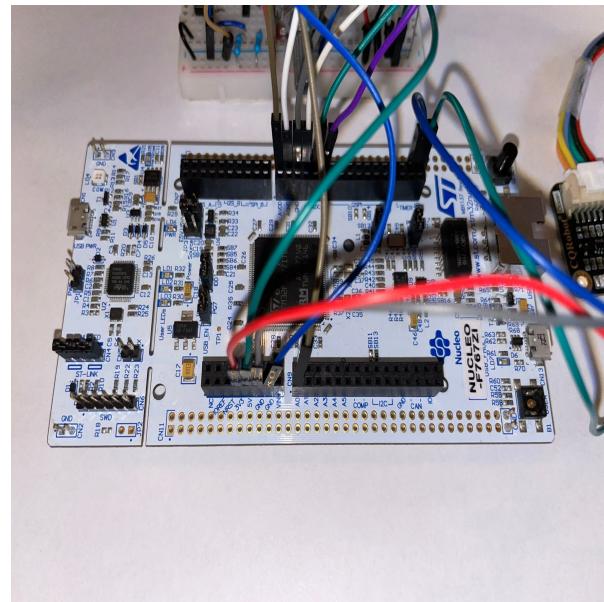




## Hardware layout



Hardware instalado



Conecciones con el board

## CubeMX parameters

Esta solución es por parte del equipo [Sensor RGB](#). Estudiantes de ingeniería en CETYS Universidad Tijuana. La siguiente tabla resume la configuración para el uso del CubeMx. Para configurarlo conlleva lo siguiente:

- System core - SysTick
- USART3 - Asynchronous

The solution uses **pointers to update the period memory location**.

Table III. Solution 1 specification

Class		Attribute	
Project	Name		F767ZI
	Clock		16Mhz
	Timebase source		SysTcl
STM32 Pins	PA3, PE0, PB1, PF13, PF14, PF15, PE9, PE11, PE13, PB10, PB11, PA14, PA13, PD9 and PB8		
FreeRTOS	Set PA3, PE0, PF13, PF14, PF15, PE9, PE11, PE13, PB10, PB11, PA14, PA13, PD9 and PB8		
Task/interrupt	Type	Name	Entry function
1	Persistent	buttonTask	buttonTaskHook
2	Interrupt		HAL_TIM_PeriodElapsedCallBack
			osPrior
			0

## ▼ Software components

## ▼ Resumen general del codigo

La funcionalidad del proyecto; usando un STM32 y un sensor capaz de leer valores RGB, es enviar datos por protocolo serial a una computadora que sirve como cliente entre un esquema cliente-servidor, para enviar por medio de TCP los datos a otra computadora que sirve como servidor y que se encarga de evaluar los datos para saber la madurez de un plátano y subirlos a una base de datos local dentro del servidor. De esta manera el mismo servidor funciona por medio de HTTP con una página que extrae los datos de la base de datos y envía datos por medio del protocolo USART hacia el STM32 con un display que muestra el valor evaluado de la base de datos.

"client.py"

```
from serial import Serial
from time import sleep
import requests
import socket
import sys

#PORT DEFINITION
stm32_port = Serial('/dev/cu.usbmodem1103', 115200, timeout=1000)

#SERVER DEFINITION
host = "10.0.0.23"
server_port = 5001

#API URL
url = "http://10.0.0.23:3000/data"
```

```
#VARIABLES
count = 0
dataSent = False

#
while True:
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect((host, server_port)) # connect to server
        print("connected!")

        while (count < 15): # for 15 rgb values
            if (stm32_port.in_waiting > 0): # wait for message from stm32
                data = stm32_port.read_until()
                data = data.decode("utf-8")

                s.sendall(data) # send to server
                print(data)

            count += 1

            dataSent = True
            count = 0

    except:
        print(sys.exc_info())
        break

    finally:
        if dataSent:
            s.close() # close connection to server
            print("closed!")

        sleep(1)

        response = requests.get(url) # get grade from database
        RGBid = response.json()[0]['idSensorGrades']
        grade = response.json()[0]['char_level']

        stm32_port.write(grade.encode()) # send grade to stm32
        print("sent " + grade + " from ID " + str(RGBid))

        dataSent = False # reset flag

"server.py"

import socket

HOST = '10.0.0.20'
PORT = 5001
```

```
BUFFER_SIZE = 1024

class Server:
    def __init__(self):
        self.host = HOST
        self.port = PORT
        self.buffer = BUFFER_SIZE
        self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server.bind((self.host, self.port))
        self.conn = None
        self.addr = None
        self.messages = []

    def up(self):
        self.server.listen(100)
        print('Server up')

    def connect(self):
        self.conn, self.addr = self.server.accept()
        print(f'Connected to: {self.addr}')

    def clientthread(self):
        while True:
            message = self.conn.recv(self.buffer)
            if message:
                if not len(message) > 14:
                    self.messages.append(message)
                #if len(self.messages) > 20:
                #    self.conn.close()
            else:
                break
        #self.conn.close()

    def handlebyte(self):
        cleaned_strings = [msg.decode('utf-8').strip() for msg in self.messages]
        self.messages = cleaned_strings

    def close():
        server.shutdown(socket.SHUT_RDWR)
        server.close()
        print ("closed")

if __name__ == '__main__':
    server = Server()
    server.up()
    while True:
        server.connect()
        server.clientthread()\
```

```
server = Client('colab')  
server.messages[-1]
```

## ▼ Código para sensor RGB

En este código se evalúa el color de la fruta y de esta manera se arroja al final un mensaje que determina si la fruta se considera madura, si está por madurar, si está pasado de maduro o si ya no es comestible. table.png

```
import datetime  
  
# DATETIME  
now = datetime.datetime.today()  
f = '%Y-%m-%d %H:%M:%S'  
now = now.strftime(f)  
  
class RGB(): # define RGB class  
    def __init__(self,r,g,b):  
        self.r = r  
        self.g = g  
        self.b = b  
        self.grade = None  
        self.chargrade = None  
  
    def dictgrade(self): # return RGB values and maturity  
        return {  
            'R': self.r,  
            'G': self.g,  
            'B': self.b,  
            'Grade': self.grade,  
            'Maturity': self.chargrade,  
            'Moment' : now  
        }  
  
    def evaluate(self): # evaluate banana maturity from RGB value (NOT_USED)  
        if ((self.r > self.g) and ((self.r - self.g) < 9)) and (self.g >= 142):  
            self.grade = 3  
            self.chargrade = 'RIPE'  
        elif ((self.r > self.g) and ((self.r-self.g) >= 9)) and (self.g >= 142):  
            self.grade = 5  
            self.chargrade = 'OVERRIPE'  
        elif (self.r > self.g) and (self.g <= 142):  
            self.grade = 7  
            self.chargrade = 'THROW AWAY'
```

```

        elif (self.g > self.b):
            self.grade = 1
            self.chargrade = 'NOT RIPE'
        return self.grade

    def evaluate_banana(self): # evaluate banana maturity from RGB value
        green_threshold = 100
        yellow_threshold = 200

        if self.g > self.r and self.g > self.b and self.g > green_threshold:
            self.grade = 1
            self.chargrade = 'NOT RIPE'
        elif self.r >= self.g >= self.b and self.g >= green_threshold and self.g < yellow_threshold:
            self.grade = 3
            self.chargrade = 'RIPE'
        elif self.r >= self.g >= self.b and self.g >= yellow_threshold:
            self.grade = 5
            self.chargrade = 'OVERRIPE'
        else:
            self.grade = 7
            self.chargrade = 'THROW AWAY'

    return self.grade

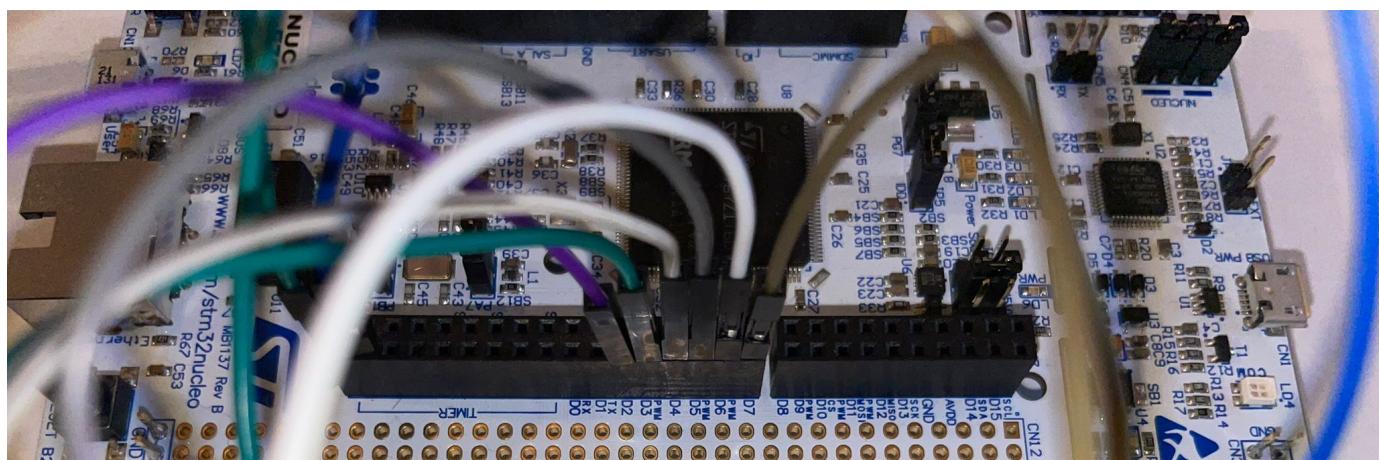
if __name__ == '__main__':
    banana = RGB(105,240,187)
    banana.evaluate()
    print(banana.grade)
    print(banana.dictgrade())

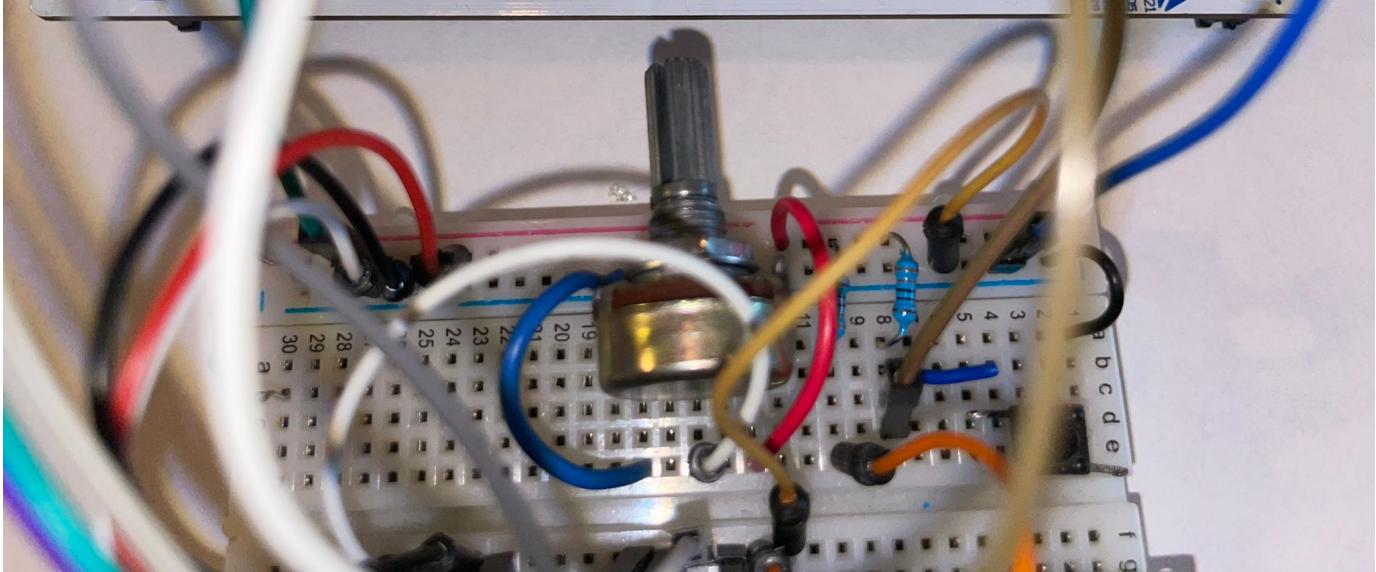
```

## ✓ Creating the message queue

Al display LCD, se conectan al STM32 de la siguiente manera: el pin RS para activar la escritura de comandos al pin PF15, del EN para activar escritura al pin E13, para transmisión de datos se utilizan los pines D4 al PF14, D5 al PE11, D6 al PE9 y el D7 al PF13.

...





El archivo main.py implementa los archivos de server.py y rgb.py, donde abre el servidor socket y abre conexiones al cliente. Cuando recibe una serie de datos del cliente (en este caso siendo los distintos valores RGB) lo almacena dentro de un objeto de la clase Series de la biblioteca de pandas. Desde este punto emplea el método mode() para encontrar el valor RGB más repetido, evalua el valor con evaluate\_banana() y lo sube a la base de datos.

```
import mysql.connector
from server import Server
from rgb import RGB
import pandas as pd

conn = mysql.connector.connect( # connect to MySQL DB
    user = 'app_usr',
    password = '7@Rvopcpwc',
    host = 'localhost',
    database = 'sys'
)

pool = conn.cursor() # set DB cursor

def process_rgb(value): # round RGB values up or down
    rgb_list = value.split(',')
    rounded_rgb = [str(round(int(x) / 10) * 10) for x in rgb_list]
    rounded_rgb = ', '.join(map(str, rounded_rgb))
    return rounded_rgb

def calculate_rgb_average(series): # gets the RGB value average (NOT USED)
    avg_r, avg_g, avg_b = 0, 0, 0
    total_items = len(series)
```

```
for item in series:
    r, g, b = map(int, item.split(','))
    avg_r += r
    avg_g += g
    avg_b += b

    avg_r /= total_items
    avg_g /= total_items
    avg_b /= total_items

    return [int(avg_r), int(avg_g), int(avg_b)]


def main():
    server = Server() #start socket server
    server.up()

    try:
        while True:
            server.connect() # connect and listen to client
            server.clientthread() # receive messages and append to list
            server.handlebyte() # formats data from byte

            res = pd.Series(server.messages) # pass list to Pandas Series column
            res = res.apply(process_rgb)
            print(res)

            mode = res.mode()[0] # find the RGB value with most occurences
            mode = mode.split(',')

            result = [int(i) for i in mode]
            #result = calculate_rgb_average(res)

            rgb = RGB(result[0], result[1], result[2]) # create instance of RGB class
            rgb.evaluate_banana()
            data = rgb.dictgrade()
            print(data)

            # upload result to MySQL DB
            result = pool.execute('''INSERT INTO SensorGrades (r, g, b, maturity_level, ch:
                                VALUES (%s, %s, %s, %s, %s, %s)'',
                                (data['R'],
                                 data['G'],
                                 data['B'],
                                 data['Grade'],
                                 data['Maturity'],
                                 data['Moment'])))

            conn.commit()
            server.messages = []
```

```

except KeyboardInterrupt: # handle interrupts
    print("Closing the program...")
    server.server.close()
    conn.close()
    print("Program closed.")

main()

```

## ❖ Evaluation

```

def evaluate_banana(self): # evaluate banana maturity from RGB value
    green_threshold = 100
    yellow_threshold = 200

    if self.g > self.r and self.g > self.b and self.g > green_threshold:
        self.grade = 1
        self.chargrade = 'NOT RIPE'
    elif self.r >= self.g >= self.b and self.g >= green_threshold and self.g < yellow_threshold:
        self.grade = 3
        self.chargrade = 'RIPE'
    elif self.r >= self.g >= self.b and self.g >= yellow_threshold:
        self.grade = 5
        self.chargrade = 'OVERRIPE'
    else:
        self.grade = 7
        self.chargrade = 'THROW AWAY'

    return self.grade

```

[1] Basado en el artículo realizado por Sandra, I Y Prayogi, R Damayanti and G Djoyowasito. Se utiliza la evaluación que contiene la siguiente tabla para convertirlo en un método de la clase RGB y añadir una calificación numérica además de una tipo string.

**Table 3. A database conclusion values at each banana maturity level**

Level	Database Conclusion Value	Maturity Level
1	Value $G > R$	Raw
3	Value $R > G$ , Value $(R - G) < 9$ and Value $G \geq 142$	Unripe
5	Value $R > G$ , Value $9 \leq (R - G)$ and $G \geq 142$	Ripe
7	Value $R > G$ and Value $142 \geq G$	Overripe

|

Además del metodo "evaluate", dentro del archivo main.py se encuentra una importación de la

librería pandas con el objetivo de utilizar la moda. La cual refiere al valor más utilizado dentro de las lecturas recibidas del STM32 para tener a evaluación el valor rgb que más prepondera.

El archivo de App.js pertenece al proyecto de React, el cual construye la página web donde se podrá visualizar el resultado de la última lectura del sensor. La página web contiene dos botones: uno para refrescar la pantalla y tomar la entrada más actual de la base de datos, y otro ligado a una barra de búsqueda para buscar alguna entrada de la base datos de acuerdo a su ID. Se utiliza el método fetch() de React para las llamadas http, tanto de GET como de POST.

```
import './App.css';
import React, { useState, useEffect } from 'react';

// ICONS
import { IoIosSearch } from "react-icons/io";
import { LuRefreshCw } from "react-icons/lu";

function App() {

  // HOOKS
  const [isNumeric, setIsNumeric] = useState(true);
  const [gradeId, setGradeId] = useState(0);
  const [moment, setMoment] = useState(null);
  const [rgbValue, setRgbValue] = useState([0, 0, 0]);
  const [maturityLevel, setMaturityLevel] = useState(0);
  const [fruitGrade, setFruitGrade] = useState("PENDING");

  const [inputValue, setInputValue] = useState('');

  // POST METHODS
  const handleChangeInput = (event) => {
    const temp = parseFloat(setInputValue(event.target.value));
    if (!isNaN(temp)) {
      setIsNumeric(true);
    } else {
      setIsNumeric(false);
    }
  };

  const onClickSearch = () => {
    const searchId = parseFloat(inputValue);
    if (!isNaN(searchId)) {
      setIsNumeric(true);
      console.log(searchId);
      findData(searchId);
    } else {
      setIsNumeric(false);
    }
  }

  return (
    <div>
      <h1>Fruit RGB</h1>
      <p>ID: {gradeId}</p>
      <p>Maturity Level: {maturityLevel}</p>
      <p>Color: {rgbValue}</p>
      <p>Grade: {fruitGrade}</p>
      <br/>
      <input type="text" value={inputValue} onChange={handleChangeInput}/>
      <button onClick={onClickSearch}>Search</button>
      <br/>
      <button onClick={refresh}>Refresh</button>
    </div>
  );
}

const refresh = () => {
  setGradeId(Math.floor(Math.random() * 10));
  setMaturityLevel(Math.floor(Math.random() * 10));
  setFruitGrade("PENDING");
  setRgbValue([0, 0, 0]);
  setMoment(new Date());
  setInputValue('');
}
```

```
j  
};  
  
const findData = async (searchId) => {  
  try {  
    const response = await fetch(  
      'http://10.0.0.20:3000/id', {  
        method: 'POST',  
        headers: {  
          'Content-Type': 'application/json',  
        },  
        body: JSON.stringify({  
          id : searchId  
        })  
      }  
    );  
  
    const json = await response.json();  
  
    setMoment(json[0]["moment"]);  
    setFruitGrade(json[0]["char_level"]);  
    setGradeId(json[0]["idSensorGrades"]);  
    setMaturityLevel(json[0]["maturity_level"]);  
    setRgbValue([json[0]["r"],json[0]["g"],json[0]["b"]]);  
  
  } catch (error) {  
    console.log(error.message);  
  }  
};  
  
// GET METHODS  
const onClickRefresh = () => {  
  fetchData();  
};  
  
const fetchData = async () => {  
  try {  
    const response = await fetch(  
      'http://10.0.0.20:3000/data'  
    );  
    const json = await response.json();  
  
    setMoment(json[0]["moment"]);  
    setFruitGrade(json[0]["char_level"]);  
    setGradeId(json[0]["idSensorGrades"]);  
    setMaturityLevel(json[0]["maturity_level"]);  
    setRgbValue([json[0]["r"],json[0]["g"],json[0]["b"]]);  
  
  } catch (error) {  
    console.log(error.message);  
  }  
}
```

```
};

useEffect(() => {
  fetchData();
}, []);

// RETURN
return (
  <div className="max-w-full px-24 items-center justify-center">

    <div className=" h-32 pt-10 bg-lightGray rounded-b-full text-center">
      <h1 className="text-6xl font-mono">
        <span>Fruit </span>
        <span className="text-redValue">R</span>
        <span className="text-greenValue">G</span>
        <span className="text-blueValue">B</span>
      </h1>
    </div>

    <div className='max-w-full items-center justify-center flex flex-col my-9'>
      <div className='flex flex-row justify-evenly relative'>
        <input
          className='w-56 h-10 relative border-midGray border-2 focus:outline-none rounded'
          value={inputValue}
          onChange={handleChangeInput}
          placeholder='find by ID'
        />
        <IoIosSearch className='w-6 h-6 absolute left-3 top-2 text-midGray' />
      </div>

      <button
        className='bg-midGray w-24 h-8 mt-5 rounded-full font-mono text-white text-xs for
        onClick={onClickSearch}
      >
        search
      </button>
    </div>

    <div className='flex flex-col items-center justify-center mx-5 mt-16'>
      <div className='max-w-full items-center justify-center flex flex-col font-mono'>
        <h2
          className='text-3xl'
        >
          latest banana reading
        </h2>
        <p className='text-l mt-2'>
          taken at {moment} with ID {gradeId}
        </p>
      </div>

      <div className='flex flex-row justify-center mx-5 ml-9'>
```

```
    <div className='w-96 h-32' style={{ backgroundColor: `rgb(${rgbValue.join(',')})` }}>

      <div className='flex flex-col ml-2 justify-evenly'>
        <p className='text-redValue'>{rgbValue[0]}</p>
        <p className='text-greenValue'>{rgbValue[1]}</p>
        <p className='text-blueValue'>{rgbValue[2]}</p>
      </div>
    </div>

    <div className='w-96 flex flex-row justify-evenly font-mono text-l'>
      <div className='flex flex-col justify-evenly'>
        <p className='mb-2'>GRADE:</p>
        <p>MESSAGE:</p>
      </div>

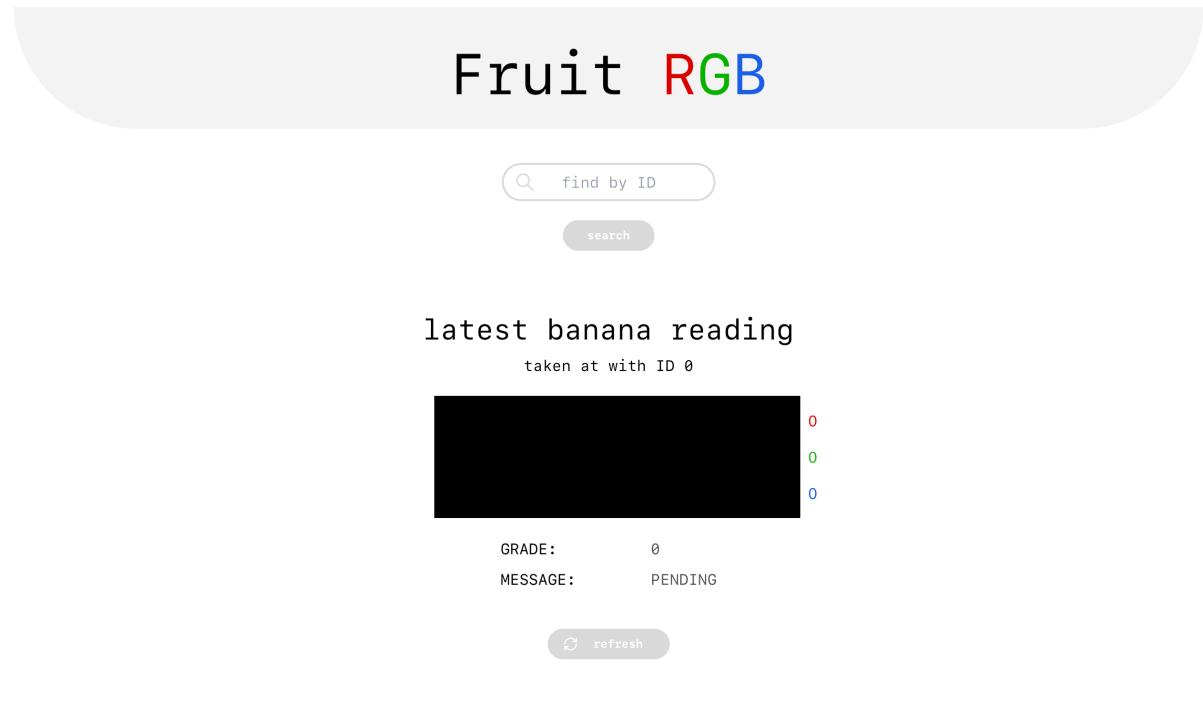
      <div className='flex flex-col justify-evenly text-darkGray'>
        <p className='mb-2'>{maturityLevel}</p>
        <p>{fruitGrade}</p>
      </div>
    </div>
  </div>

  <div className='flex flex-row justify-center'>
    <div className='w-48 flex flex-row justify-center mt-5 relative text-white'>
      <button
        className='bg-midGray w-32 h-8 mt-5 rounded-full font-mono text-xs font-bold'
        onClick={onClickRefresh}
      >
        <p className='absolute left-20 top-7'>refresh</p>
      </button>
      <LuRefreshCw className='w-4 h-4 absolute top-7 left-12' />
    </div>
  </div>
</div>
);

}

export default App;
```

## ▼ Página Web



|

## Software components

- Libreria de python - datetime
- Libreria Propia - from rgb import RGB
- Libreria Propia - from server import Server
- Libreria - import mysql.connector
- Libreria - import pandas as pd
- Libreria - tailwind css
- Libreria - react
- Código - fuente en Python

## ▼ Reference

- [1]. Sandra, I Y Prayogi, R Damayanti and G Djoyowasito, “Design to prediction tools for banana maturity based on image processing,” in IOP Conference Series: Earth and Environmental Science, vol. 475, no. 1, pp. 012010, 2020. [Online]. Available: <https://iopscience.iop.org/article/10.1088/1755-1315/475/1/012010>

