Online Herbs Detecting Simulation

Mehmet Yücel Sarıtaş - 21990451

2 January 2024

Abstract:

In the agricultural fields, it is convenient to detect and remove agricultural herbs to increase the yield of output. By detecting herbs (In my case red balls will represent tomato (maybe different balls can represent other vegetables), and green objects will represent agricultural herbs. It is just an analogy to real-world application) It is easier for the farmer to understand their agricultural field herbs situation. My project aim is to create an approach and realize this detecting mechanism using STM32, servo motors, cameras, and other electronic devices.

Youtube Video: https://youtu.be/UNbD-ZfWUSE

General Preliminary Design:

First, I will detect the position of the herb by using the camera on the PC, Subsequently, I will send this data to STM32 by using communication protocols. After Getting the right position of the herb my pan-tilt servo motor directs its position to the herb. A red laser will show the herb at the end of the application and I will also show the pixel locations (x, y) of the herb on the I2C LCD. I divided my work into two sections. The first one (Image Processing) is capturing object images and calculating coordinates of them. The second section is sending this data from the computer to STM32 through the USART communication protocol. I converted this coordinated to angles.

The Concepts and Sensors Utilized:

- Pooling
- Interrupts
- USART and I2C Communication Protocols
- Capturing PWM
- STM32 Libraries

- LCD Display
- Red Laser
- Servo Motors
- Pan Tilt Kit
- 3D printed Mount

Section 1: Image Capture and Processing Pipeline

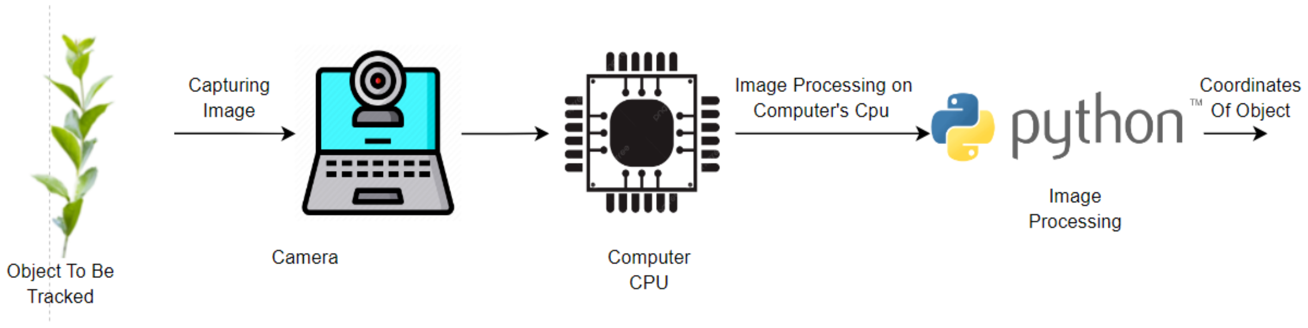
Object to Be Tracked: When a physical object is present in the area to be tracked (frame of camera), this is the beginning of the tracking process.

Camera: The camera takes images of the herb (i.e. in my case green object) online. It functions as the main sensor input, transforming the incoming visual data into processable digital data. I converted the original resolution to 600x450 in the image processing part in Python.

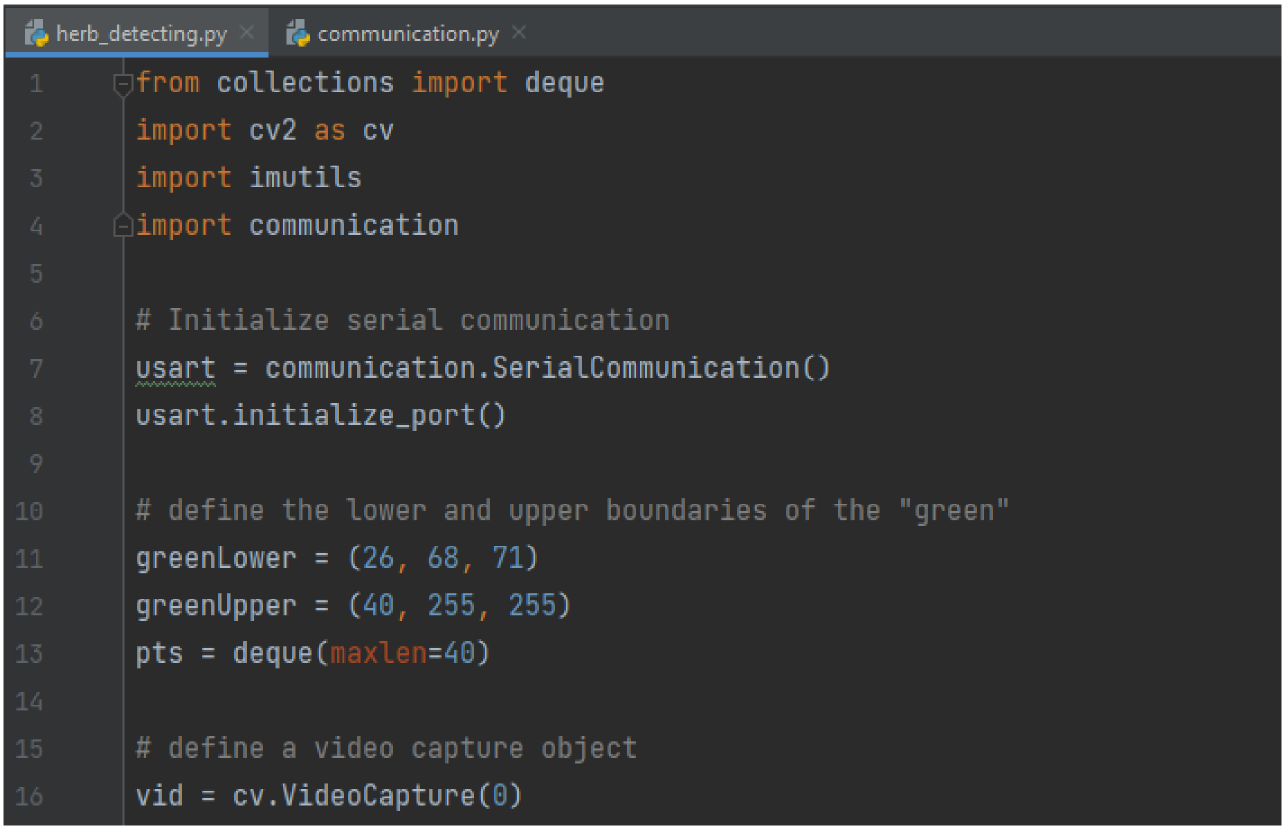
Computer CPU: A computer receives and processes the digital image that was taken by the camera. Image processing methods are carried out by the CPU (my laptop CPU), with the aid of the program Python.

Python Image Processing: This step entails applying Python programming to analyze the picture data. This may involve detecting the object, figuring out where it is in the frame, and obtaining its coordinates with the core library Open-CV.

Coordinates of Object: The coordinates of the object inside the collected frame are the outcome of the image processing. Tracking the position and movement of the object requires these coordinates.



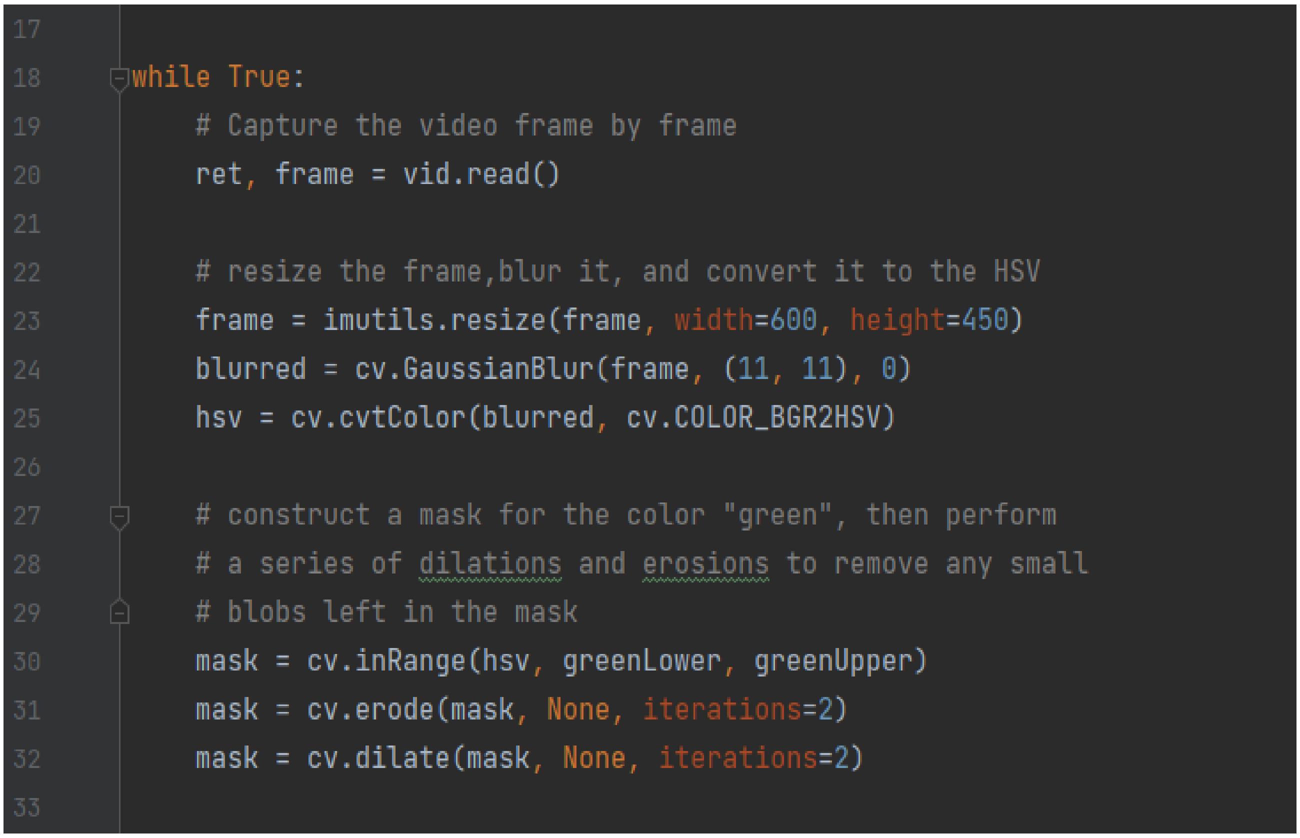
Python File of Detecting Herb (green ball)



```python
from collections import deque
import cv2 as cv
import imutils
import communication

# Initialize serial communication
usart = communication.SerialCommunication()
usart.initialize_port()

# define the lower and upper boundaries of the "green"
greenLower = (26, 68, 71)
greenUpper = (40, 255, 255)
pts = deque(maxlen=40)

# define a video capture object
vid = cv.VideoCapture(0)
```

Explanation:

- **Library Imports:** Identifies the libraries being utilized.
- **Serial Communication Initialization:** Initialize the serial communication module with SerialCommunication class that I have written.
- **Color Thresholds:** In HSV format, Masks the bottom and upper bounds of the color "green".
- **Deque Initialization:** To store the last N points, initialize a deque named pts with a maximum length of 40.
- **Video Capture Initialization:** Using OpenCV, a video capture object (vid) is created, and video is captured from the default camera (index 0).



```

while True:
    # Capture the video frame by frame
    ret, frame = vid.read()

    # resize the frame,blur it, and convert it to the HSV
    frame = imutils.resize(frame, width=600, height=450)
    blurred = cv.GaussianBlur(frame, (11, 11), 0)
    hsv = cv.cvtColor(blurred, cv.COLOR_BGR2HSV)

    # construct a mask for the color "green", then perform
    # a series of dilations and erosions to remove any small
    # blobs left in the mask
    mask = cv.inRange(hsv, greenLower, greenUpper)
    mask = cv.erode(mask, None, iterations=2)
    mask = cv.dilate(mask, None, iterations=2)

```

Explanation:

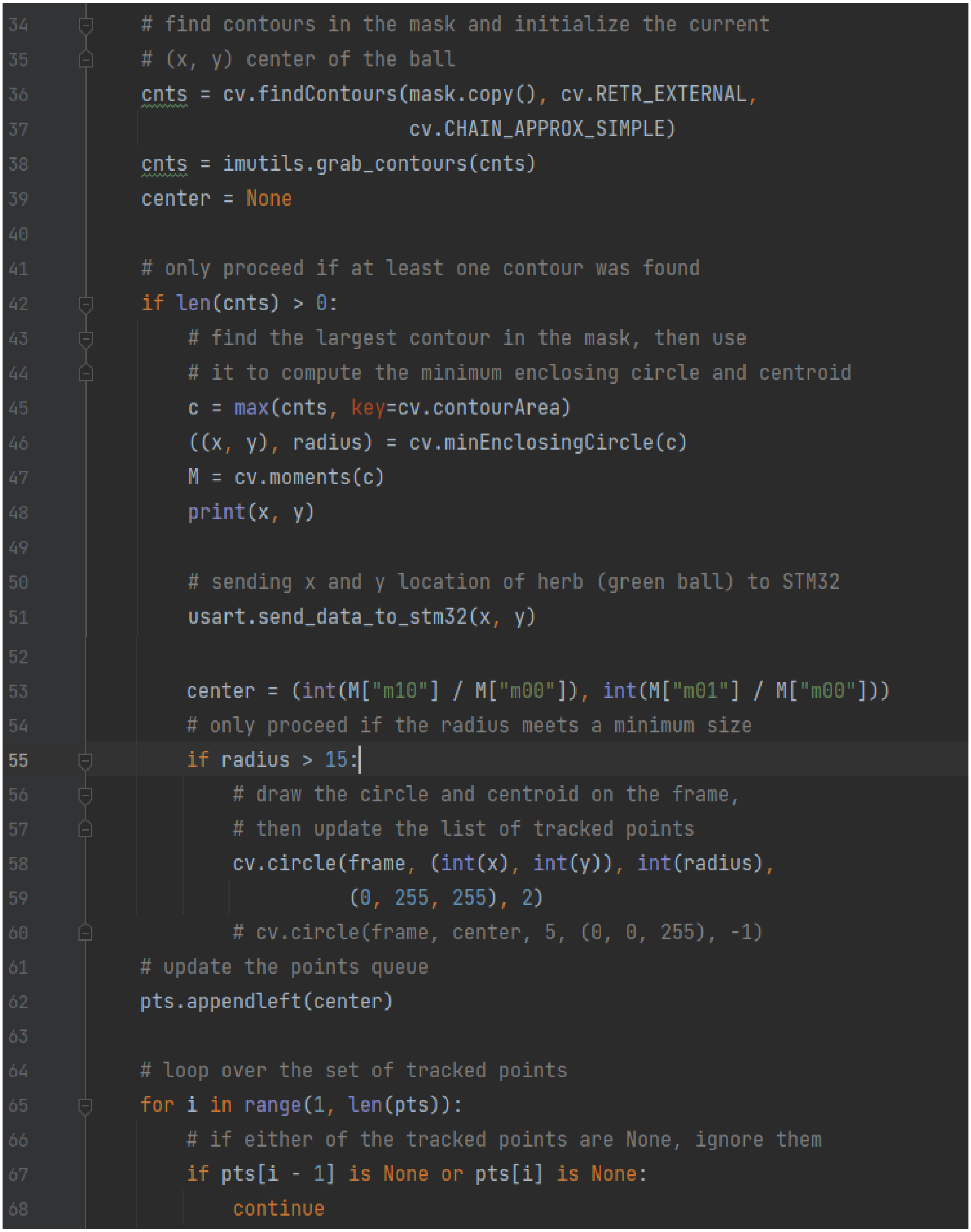
Frame Capture: Using the video capture object (vid.read()), captures the video frame by frame.

Prior to processing:

- Resize: To make processing easier, the frame is resized to 600 by 450 in width and height.
- Blur: To lessen noise, apply Gaussian blur to the frame using a kernel size of (11, 11).
- Color Space Conversion: This process changes the BGR color space of the blurry frame to HSV.

Utilizing Color Masking:

- Mask Creation: With the predefined lower and higher borders for the color "green" in HSV, a mask is created using the inRange function.



```python
# find contours in the mask and initialize the current
# (x, y) center of the ball
cnts = cv.findContours(mask.copy(), cv.RETR_EXTERNAL,
                       cv.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)
center = None

# only proceed if at least one contour was found
if len(cnts) > 0:
    # find the largest contour in the mask, then use
    # it to compute the minimum enclosing circle and centroid
    c = max(cnts, key=cv.contourArea)
    ((x, y), radius) = cv.minEnclosingCircle(c)
    M = cv.moments(c)
    print(x, y)

    # sending x and y location of herb (green ball) to STM32
    usart.send_data_to_stm32(x, y)

    center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
    # only proceed if the radius meets a minimum size
    if radius > 15:
        # draw the circle and centroid on the frame,
        # then update the list of tracked points
        cv.circle(frame, (int(x), int(y)), int(radius),
                  (0, 255, 255), 2)
        # cv.circle(frame, center, 5, (0, 0, 255), -1)
# update the points queue
pts.appendleft(center)

# loop over the set of tracked points
for i in range(1, len(pts)):
    # if either of the tracked points are None, ignore them
    if pts[i - 1] is None or pts[i] is None:
        continue
```

Explanation:

Contour Detection:

- Locate Contours: Locates contours in the mask using cv.findContours.

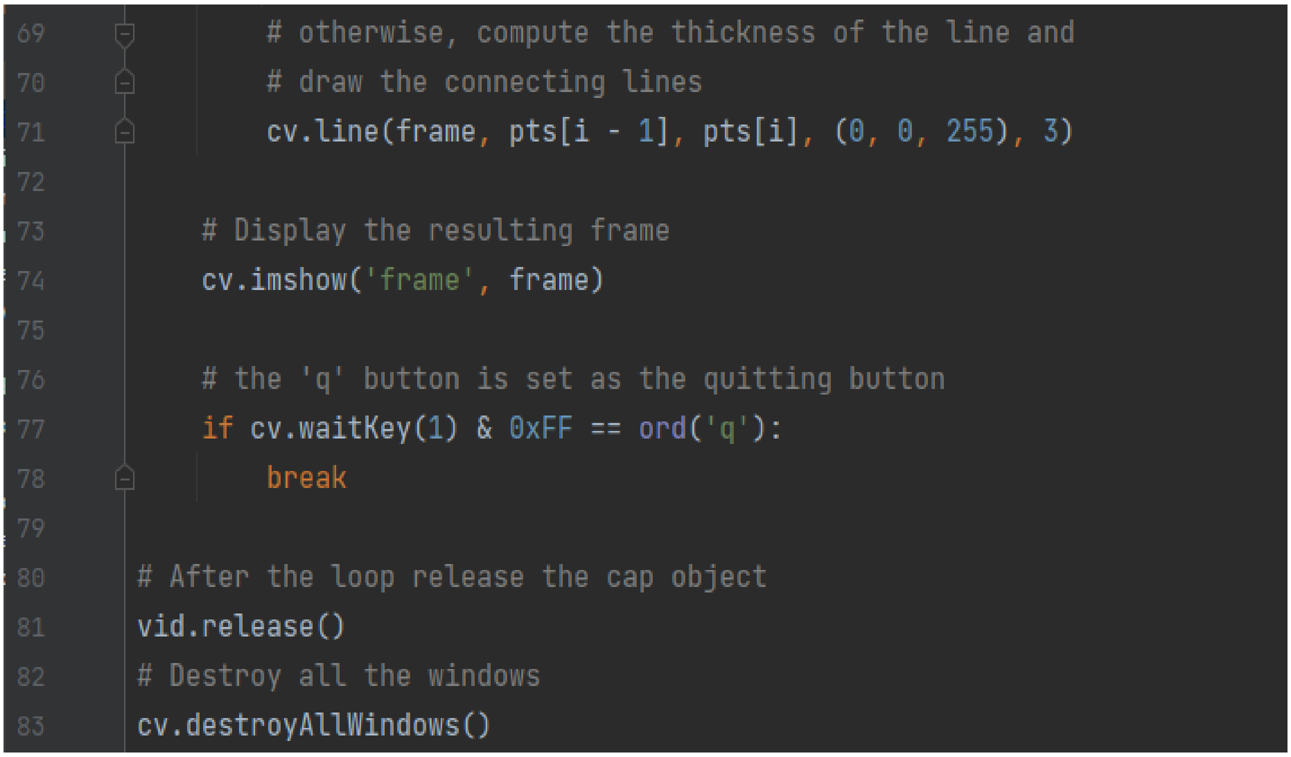
Processing of Contours:

- Largest Contour: Locates the mask's largest contour. The greatest contour's minimum enclosing circle and centroid are calculated using the circle and centroid formula.

Sending Data: Using usart.send_data_to_stm32(x, y), sends the herb's x and y location to the STM32.

Illustration:

- Centroid and Circle Drawing: On the original frame, draw a circle encircling the herb and its centroid.
- Points Queue Update: Adds the centroid to the points queue (pts).



```
            # otherwise, compute the thickness of the line and
            # draw the connecting lines
            cv.line(frame, pts[i - 1], pts[i], (0, 0, 255), 3)

        # Display the resulting frame
        cv.imshow('frame', frame)

        # the 'q' button is set as the quitting button
        if cv.waitKey(1) & 0xFF == ord('q'):
            break

    # After the loop release the cap object
    vid.release()
    # Destroy all the windows
    cv.destroyAllWindows()
```

Explanation: Simply terminates the all image processing operations.

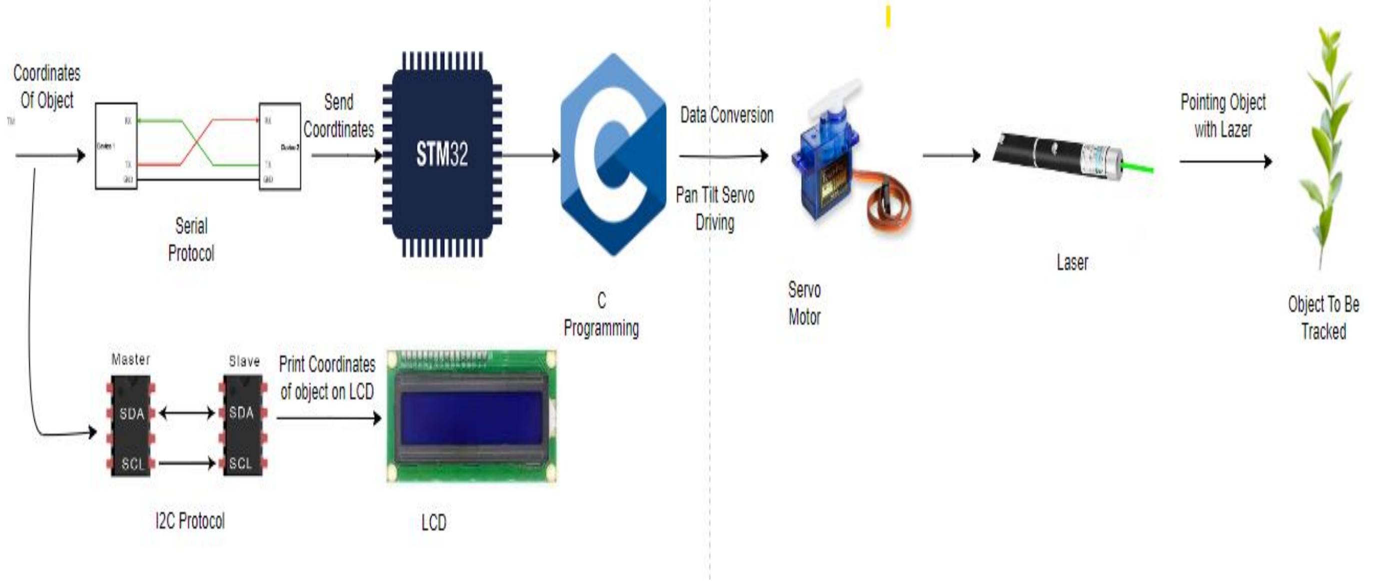
Section 2: Data Handling and Actuation Pipeline

Coordinates of Object: This is the data that was obtained during the image processing phase and it shows where the object is. It is in pixel format. However, I design an algorithm to convert pixels to angle.

Serial Protocol: To interface with microcontrollers or other processing units, the coordinates are communicated over a serial communication (USART) protocol. To be able to send data from my computer to STM32 I have used Interrupt. Because I don't want to introduce another delay to the system by data manipulation. So I have preferred using Interrupts rather than the Pooling method.

STM32 and C Programming: STM32 microcontroller, which gets the coordinates of the item. All the processes rather than image processing are handled inside of STM32 in my project. I have used C programming languages and libraries to make my hardware useful.

I2C Protocol: To display the coordinates of an object, an LCD (acting as the slave) and an STM32 (acting as the master) communicate via the

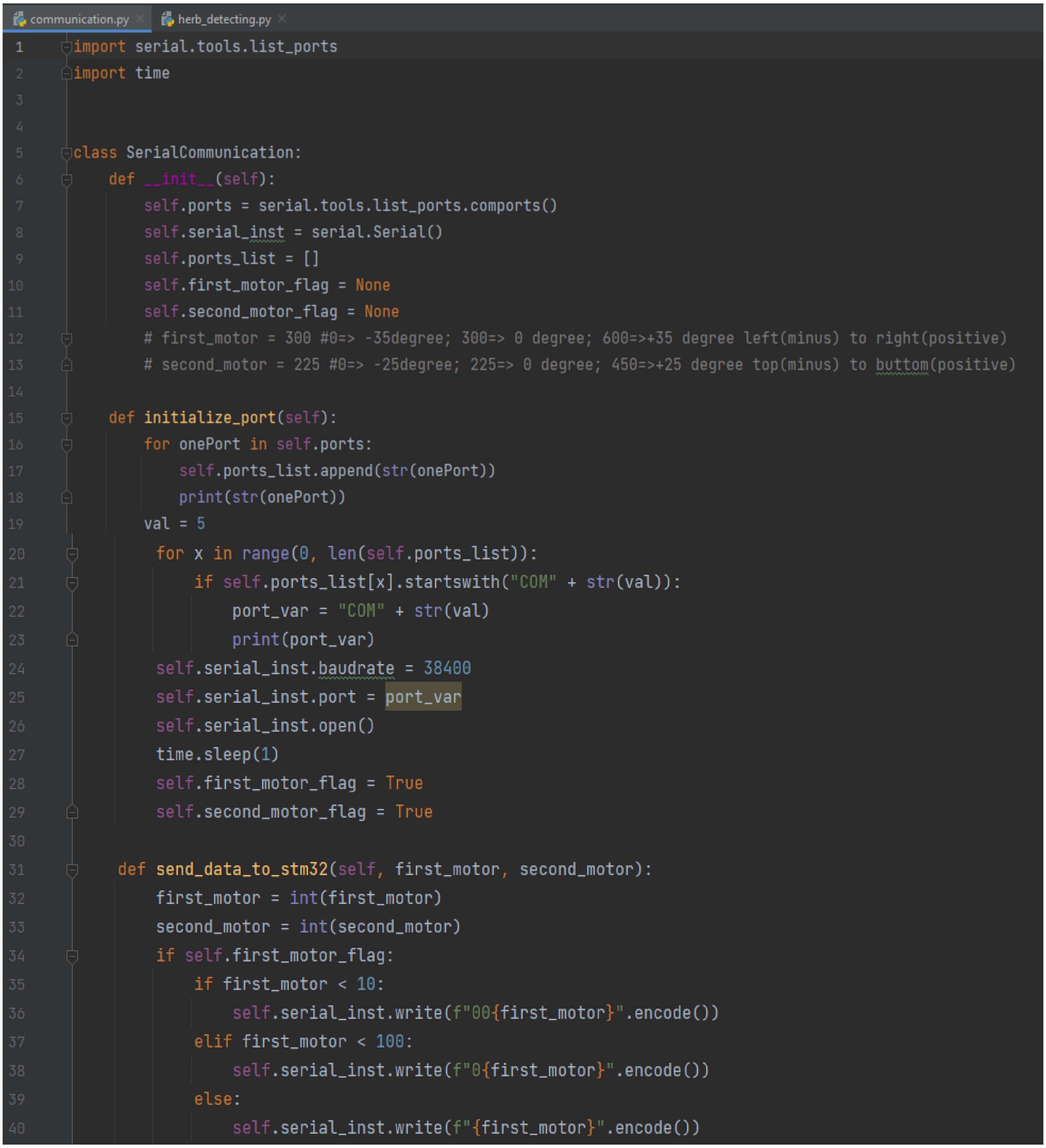


LCD: A Liquid Crystal Display, which provides a user interface for tracking process monitoring and displays the coordinates of the item. I have shown my coordinates in pixels on LCD. It mainly shows the x position and y position with scales 600 and 450 respectively.

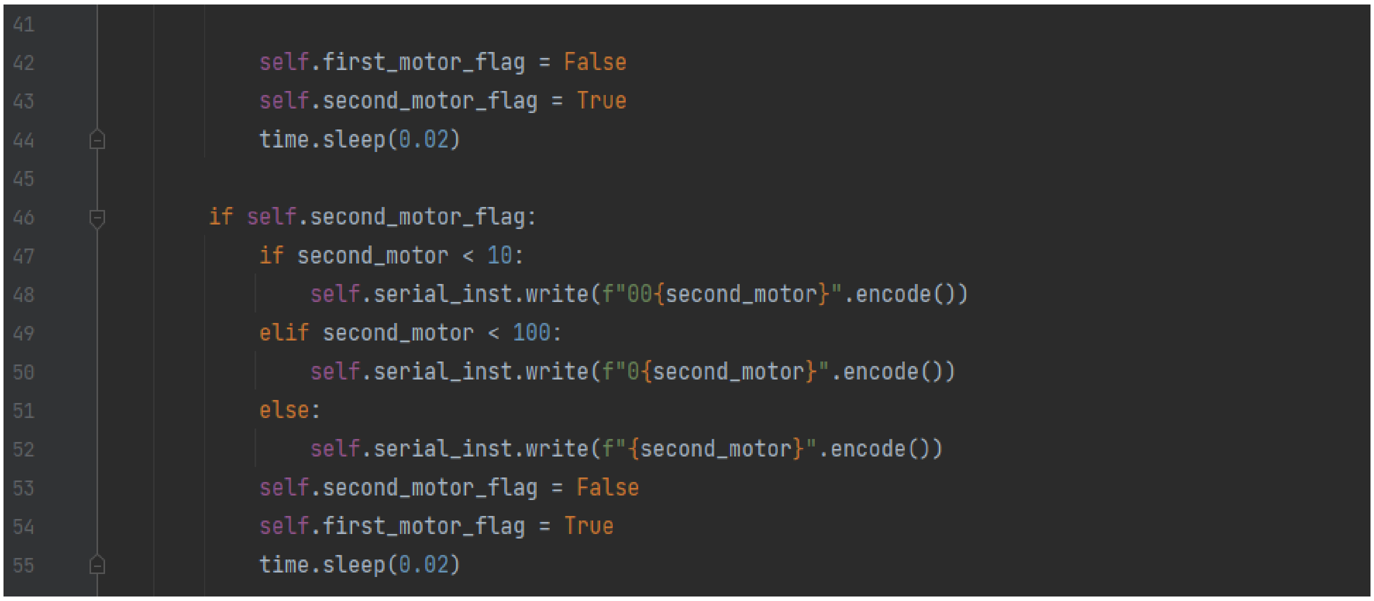
Pan Tilt Servo Driving and Data Conversion: The motors position the laser by adjusting their angles in response to signals derived from the processed data. The raw data (coordinates) are transformed into a format (angle i.e. yaw and pitch) that can be used to operate servo motors, which will change the laser's location in 2 axes.

Laser: The object being tracked is pointed at with the laser to provide a visual indication of its position. This last action brings the interaction cycle to a close by precisely pointing the laser at the tracked item.

Python File of Sending Coordinates of Herb



```python
import serial.tools.list_ports
import time


class SerialCommunication:
    def __init__(self):
        self.ports = serial.tools.list_ports.comports()
        self.serial_inst = serial.Serial()
        self.ports_list = []
        self.first_motor_flag = None
        self.second_motor_flag = None
        # first_motor = 300 #0=> -35degree; 300=> 0 degree; 600=>+35 degree left(minus) to right(positive)
        # second_motor = 225 #0=> -25degree; 225=> 0 degree; 450=>+25 degree top(minus) to buttom(positive)

    def initialize_port(self):
        for onePort in self.ports:
            self.ports_list.append(str(onePort))
            print(str(onePort))
        val = 5
        for x in range(0, len(self.ports_list)):
            if self.ports_list[x].startswith("COM" + str(val)):
                port_var = "COM" + str(val)
                print(port_var)
        self.serial_inst.baudrate = 38400
        self.serial_inst.port = port_var
        self.serial_inst.open()
        time.sleep(1)
        self.first_motor_flag = True
        self.second_motor_flag = True

    def send_data_to_stm32(self, first_motor, second_motor):
        first_motor = int(first_motor)
        second_motor = int(second_motor)
        if self.first_motor_flag:
            if first_motor < 10:
                self.serial_inst.write(f"00{first_motor}".encode())
            elif first_motor < 100:
                self.serial_inst.write(f"0{first_motor}".encode())
            else:
                self.serial_inst.write(f"{first_motor}".encode())
```



```
            self.first_motor_flag = False
            self.second_motor_flag = True
            time.sleep(0.02)

        if self.second_motor_flag:
            if second_motor < 10:
                self.serial_inst.write(f"00{second_motor}".encode())
            elif second_motor < 100:
                self.serial_inst.write(f"0{second_motor}".encode())
            else:
                self.serial_inst.write(f"{second_motor}".encode())
            self.second_motor_flag = False
            self.first_motor_flag = True
            time.sleep(0.02)
```

Explanation:

Initialization of the class:

The __init__ method initializes a number of properties required for serial communication, such as a serial instance, information about accessible ports (COM5 in my case), and motor control flags.

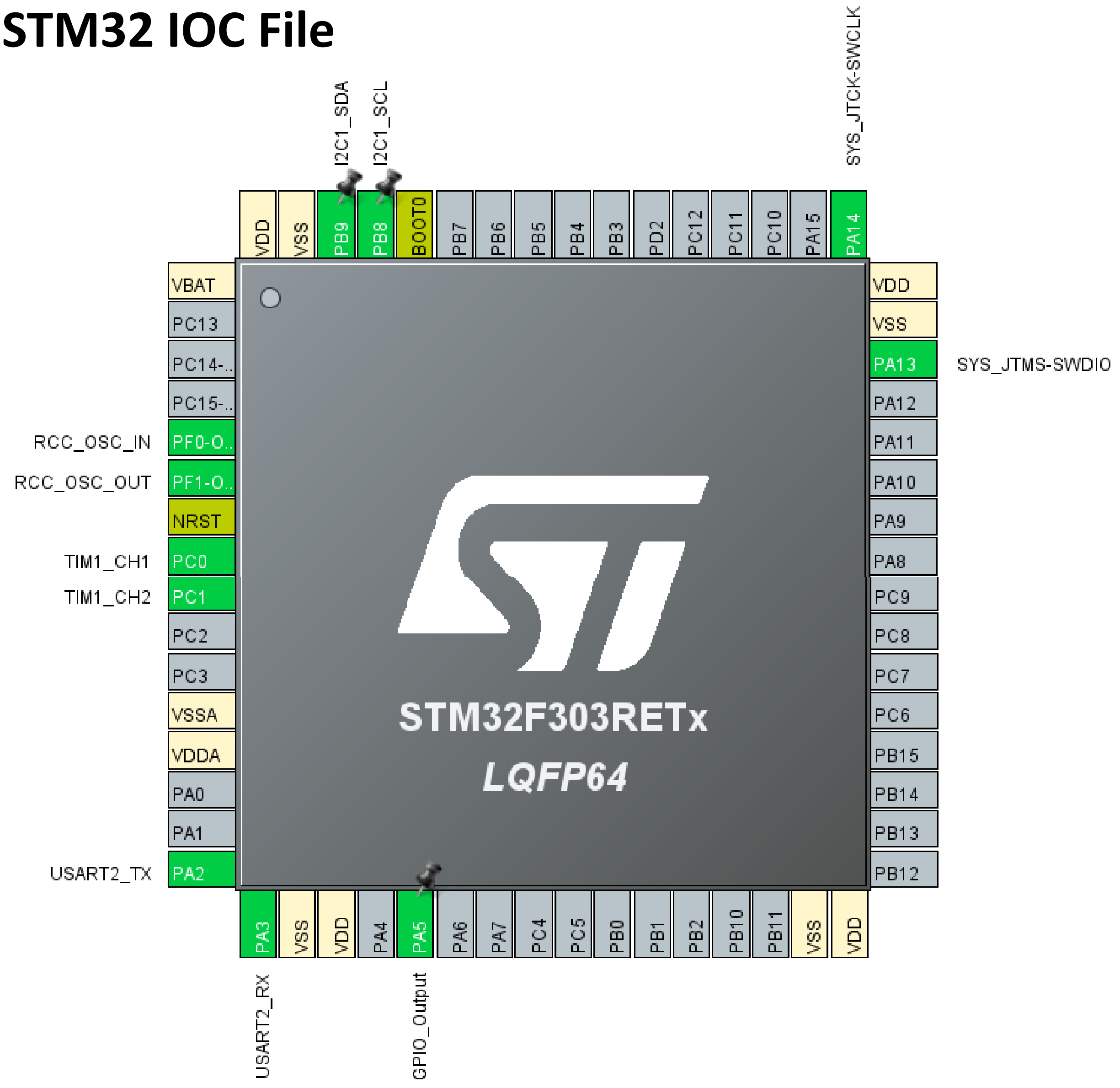
Initialization of the port:

The initialize_port function lists all available ports and chooses the one that begins with COM5. Next, it opens the port, initializes flags, and configures the serial communication parameters.

Data Transfer (method send_data_to_stm32):

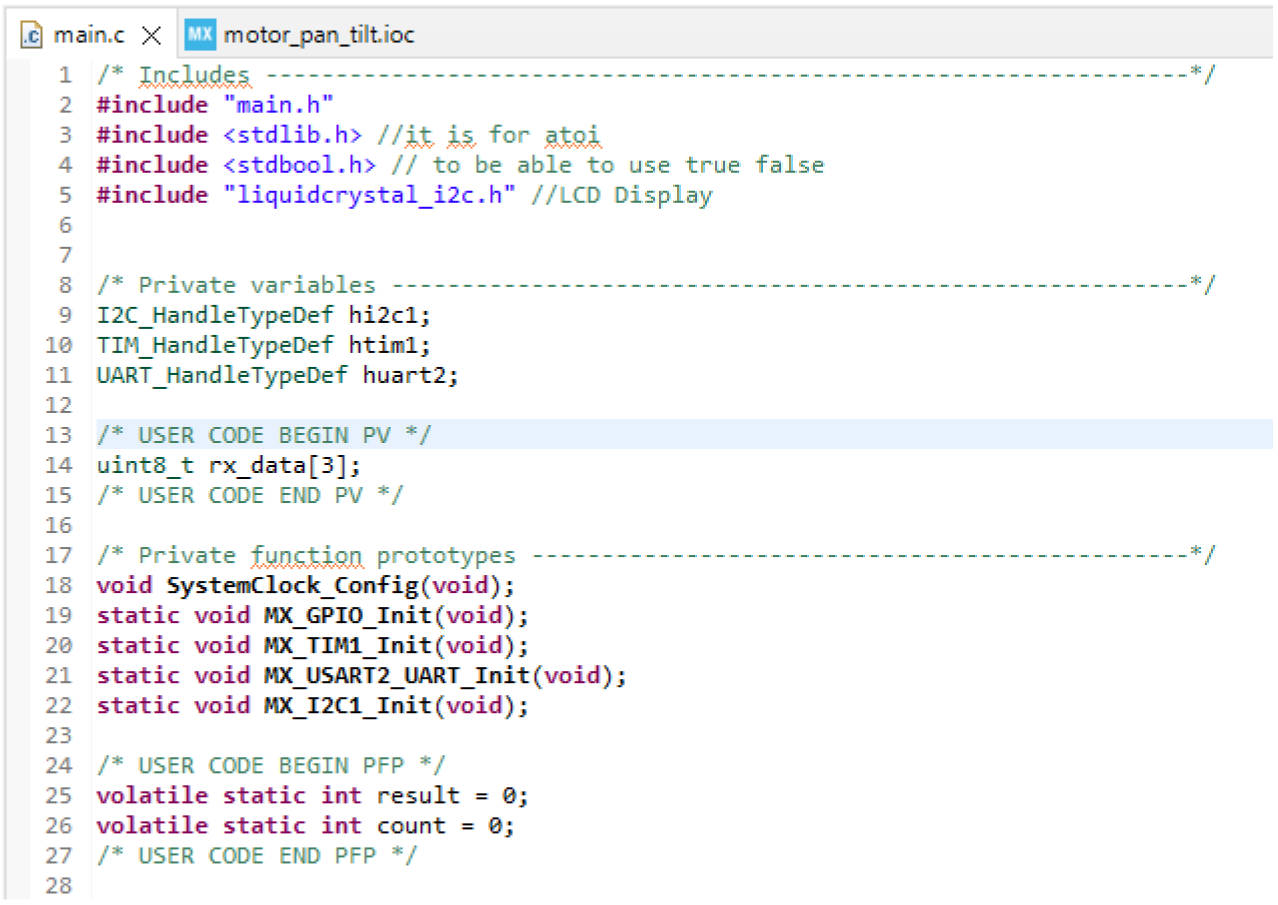
The Coordinate data that is sent to the STM32 is handled by this procedure. Two Coordinates values are required as parameters. To guarantee correct handling, the motor values are transformed to integers. Depending on the status of the flags (first_motor_flag and second_motor_flag), data for the first and second motors is given to the STM32. Since the usart interrupt handle datas circular I added ‘00’ for data lowe than 10, ‘0’ lower than 100 to ensure data is tx and rx correctly.

STM32 IOC File



TIM_CH1: To drive first motor (yaw angle)
TIM_CH2: To drive second motor (pitch angle)
USART2: To communicate between STM32 and Laptop
I2C1: LCD manupulation
RCC: System clock

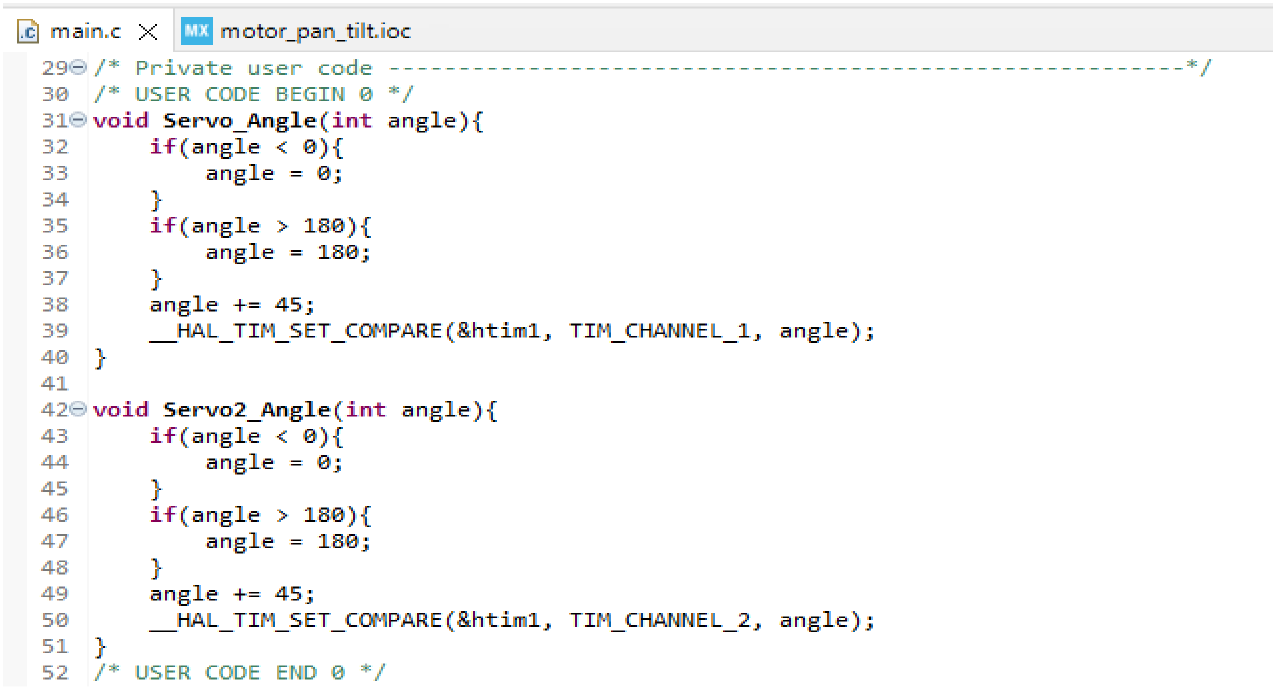
C main File In My STM32


main.c motor_pan_tilt.ioc

```c
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include <stdlib.h> //it is for atoi
#include <stdbool.h> // to be able to use true false
#include "liquidcrystal_i2c.h" //LCD Display


/* Private variables ---------------------------------------------------------*/
I2C_HandleTypeDef hi2c1;
TIM_HandleTypeDef htim1;
UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */
uint8_t rx_data[3];
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM1_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_I2C1_Init(void);

/* USER CODE BEGIN PFP */
volatile static int result = 0;
volatile static int count = 0;
/* USER CODE END PFP */

```

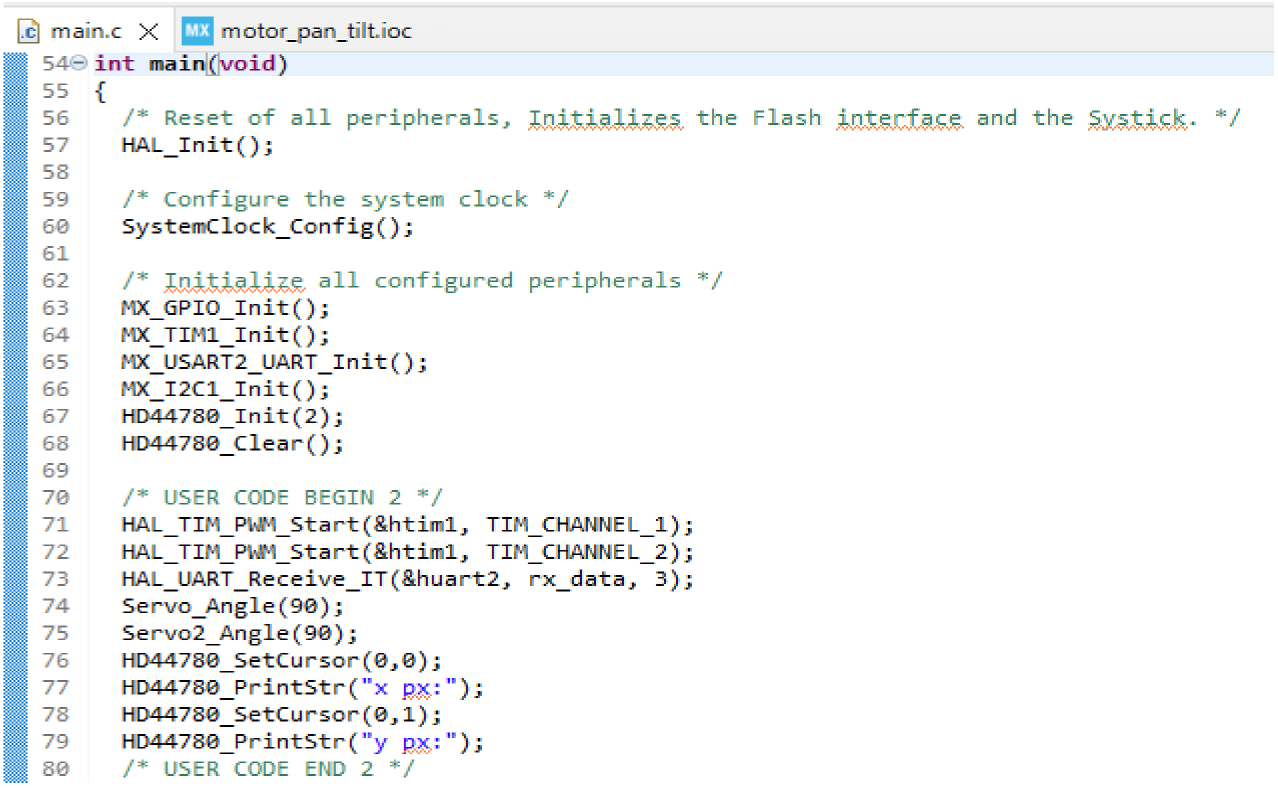
Here, there are STM32 CUBE IDE ioc initializations and variables assignments..

- I have used **count** variable for 2 purposes. First, to select which motor to receive sended data, Second, to down sample data printed on LCD display. I will show this algorithm in USART interrupt section.
- **rx_data[3]** array is used for holding sending data as **char**.
- **result** variable is used for holding sended data as **int.**



main.c | motor_pan_tilt.ioc

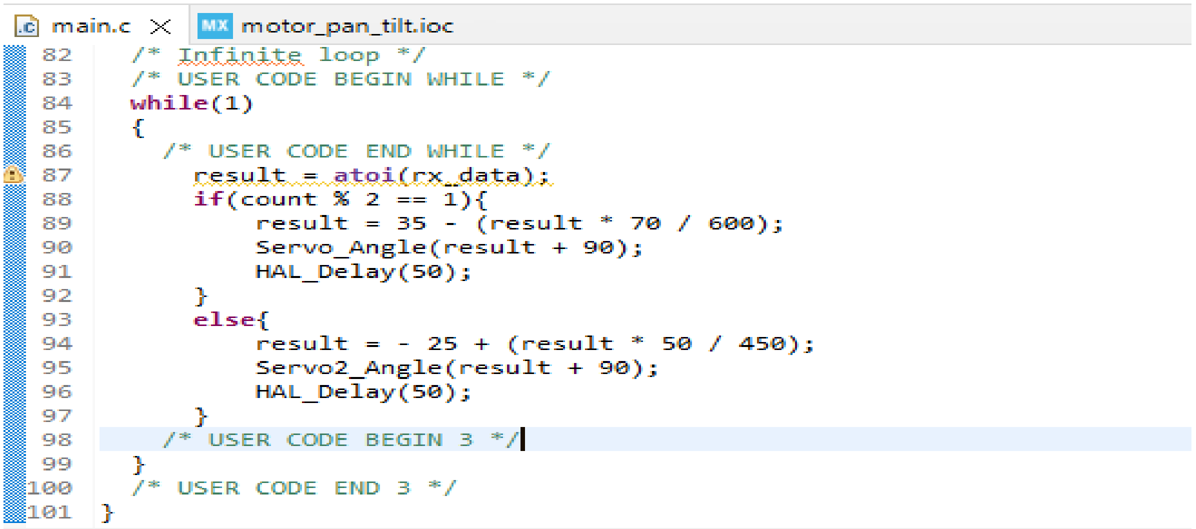
```
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
void Servo_Angle(int angle){
    if(angle < 0){
        angle = 0;
    }
    if(angle > 180){
        angle = 180;
    }
    angle += 45;
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, angle);
}

void Servo2_Angle(int angle){
    if(angle < 0){
        angle = 0;
    }
    if(angle > 180){
        angle = 180;
    }
    angle += 45;
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, angle);
}
/* USER CODE END 0 */
```



main.c | motor_pan_tilt.ioc

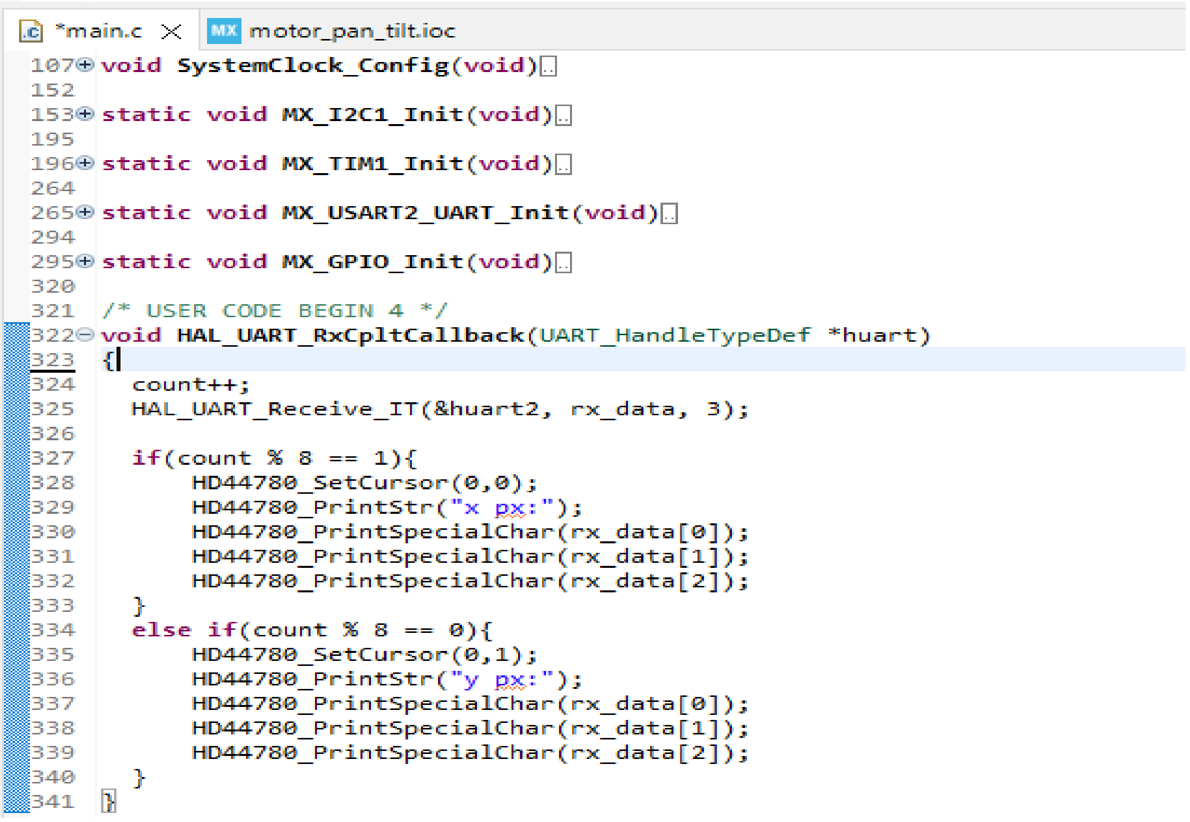
```
int main(void)
{
  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM1_Init();
  MX_USART2_UART_Init();
  MX_I2C1_Init();
  HD44780_Init(2);
  HD44780_Clear();

  /* USER CODE BEGIN 2 */
  HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
  HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);
  HAL_UART_Receive_IT(&huart2, rx_data, 3);
  Servo_Angle(90);
  Servo2_Angle(90);
  HD44780_SetCursor(0,0);
  HD44780_PrintStr("x px:");
  HD44780_SetCursor(0,1);
  HD44780_PrintStr("y px:");
  /* USER CODE END 2 */
```

Servo_Angle and **Servo2_Angle** function respectively.
HD44780_ is library method of liquidcrystal_i2c.h library for handling LCD.
Other functions are again STM32 CUBE IDE ioc initializations



main.c ✕ | MX motor_pan_tilt.ioc

```
    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while(1)
    {
      /* USER CODE END WHILE */
        result = atoi(rx_data);
        if(count % 2 == 1){
            result = 35 - (result * 70 / 600);
            Servo_Angle(result + 90);
            HAL_Delay(50);
        }
        else{
            result = - 25 + (result * 50 / 450);
            Servo2_Angle(result + 90);
            HAL_Delay(50);
        }
      /* USER CODE BEGIN 3 */
    }
    /* USER CODE END 3 */
  }
```



*main.c ✕ | MX motor_pan_tilt.ioc

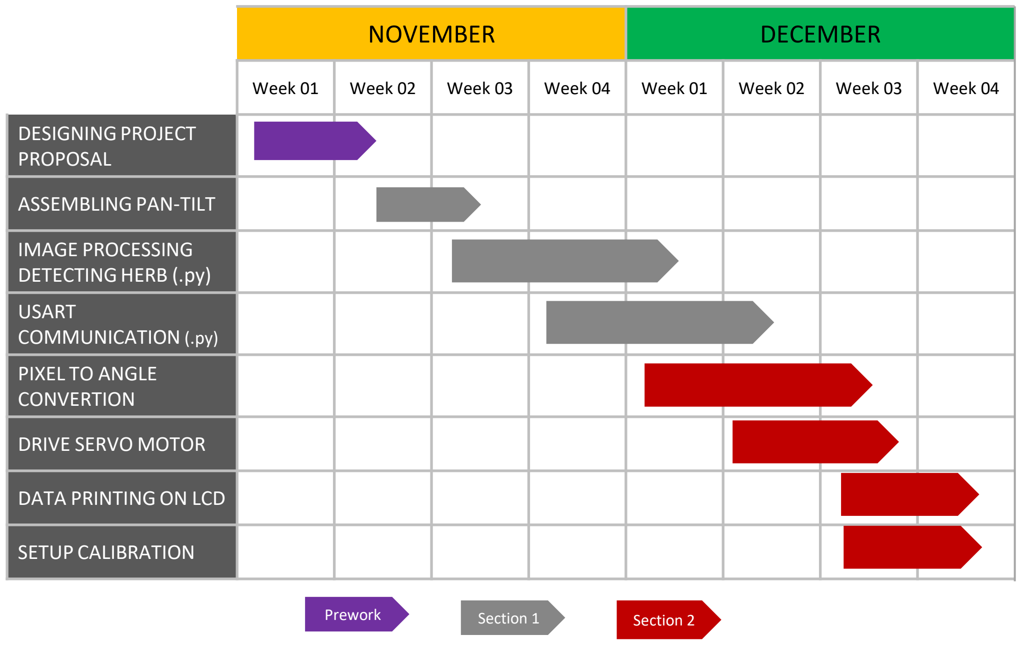
```
void SystemClock_Config(void)

static void MX_I2C1_Init(void)

static void MX_TIM1_Init(void)

static void MX_USART2_UART_Init(void)

static void MX_GPIO_Init(void)

/* USER CODE BEGIN 4 */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    count++;
    HAL_UART_Receive_IT(&huart2, rx_data, 3);

    if(count % 8 == 1){
        HD44780_SetCursor(0,0);
        HD44780_PrintStr("x px:");
        HD44780_PrintSpecialChar(rx_data[0]);
        HD44780_PrintSpecialChar(rx_data[1]);
        HD44780_PrintSpecialChar(rx_data[2]);
    }
    else if(count % 8 == 0){
        HD44780_SetCursor(0,1);
        HD44780_PrintStr("y px:");
        HD44780_PrintSpecialChar(rx_data[0]);
        HD44780_PrintSpecialChar(rx_data[1]);
        HD44780_PrintSpecialChar(rx_data[2]);
    }
}
```

My Interrupt routine(**HAL_UART_RxCpltCallback**) has three specific duties.

- Incrementing ‘count’ (Usage of it has already been explained)
- Receiving data sent from the computer.
- Printing coming data to LCD (downsampled)

Inside of **while(1)** I simply convert pixels to yaw and pitch angles and drive motors concerning those angles. The conversion algorithm may change for different cameras.

GANNT CHART



Note: All work is done by myself.
Student1 : Mehmet Yücel Sarıtaş -> Percentage Contribution %100

Conclusion: Together with two sections explain my system which takes an object's picture, processes it to find its location, and then utilizes that information to aim a laser at it. My project might be a component of a bigger automated interaction and object-tracking system used in robotics or automated surveillance.

Youtube Video: https://youtu.be/UNbD-ZfWUSE