HACETTEPE UNIVERSITY DEPARTMENT

OF COMPUTER ENGINEERING

BBM-434

Project Proposal

RGB COLOR TRACKING CAMERA

Group: gigadeth
Teammate 1: Çağdaş Utku Tonbul, 21427468
Teammate 2: Furkan Karakökçek, 21328155

Contents

1 Introduction

1.1 Overview

The scope of the project is to develop RGB color tracking camera. This device would track one of the RGB color by understanding the direction of the color by looking RGB intensity matrix of image and servo motors would allow the device to track the color.

Initially our server would periodically analyze frames from video feed of camera in order to differentiate RGB color direction from its color intensity matrix.Arduino Uno selected for capturing images. Secondly, server would send UDP packets to our launchpad that ESP8266 wifi module would capture. This UDP packet would consist of information that only red values of frames.This approach provided to compress the packet size and hiding information . Finally, launchpad tracks color by moving servo motors to its corresponding degree such as 30.

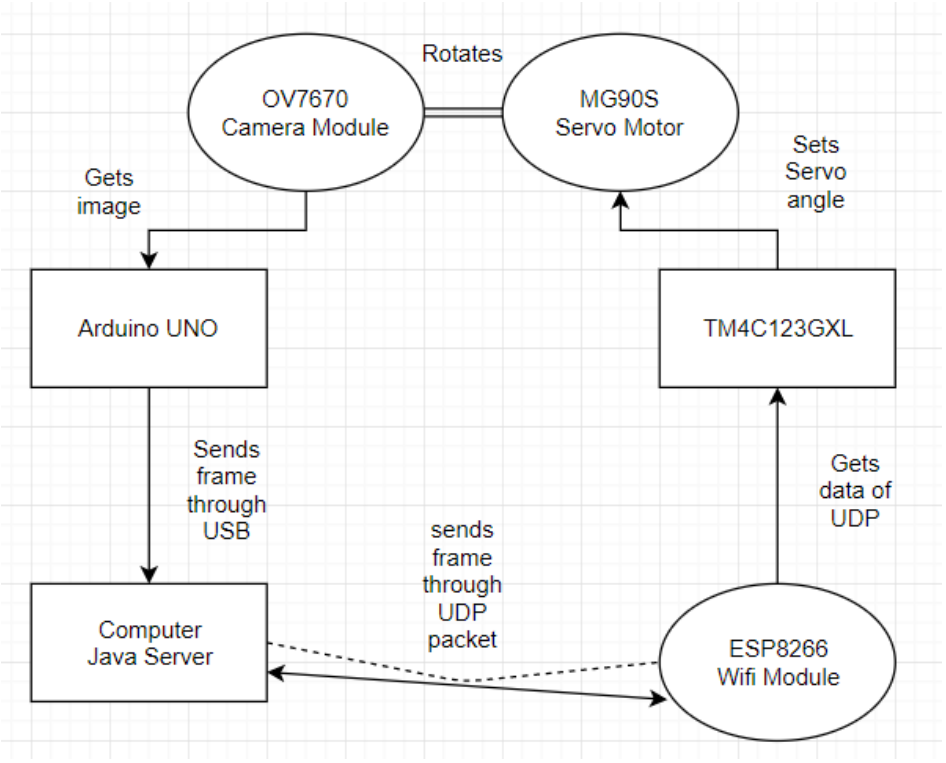


Figure 1: General view of project

1.2 Outline

This report contains final decisions, view, works done of project. The report wraps up with some photographs of the RGB color tracking camera. The hardware semantics and the steps to rebuild the project from the attached code repository ”projectcodes.zip” are relegated to appendices.

2 Real-time design consideration

2.1 Real-time tasks

We used this functions;
SetServoAngle(angle): It sets the current angle of the serve.
Delay100ms(ms): While capturing the frame, each frame first must be processed then angle was set.
ESP8266_MakeTCPConnection('IP'): This opens the socket in Java-server.
ESP8266_SendTCP(Frame):Sends the frames to appropriate socket.

2.2 State-machine design approach

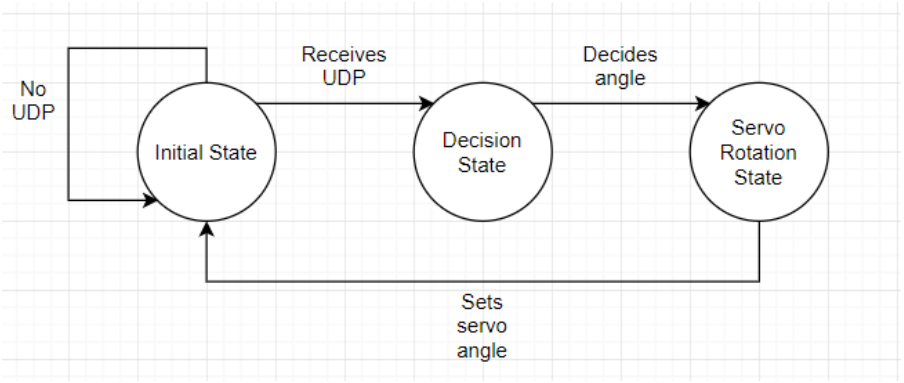


Figure 2: Finite state machine design of project

3 Project Features

3.1 Hardware System

There will be external camera, ESP8266 Wifi module, Arduino UNO and one servo in our project. We will periodically take picture from our camera and due to our launchpad's 32 kb Ram and our launchpad doesn't have Serial Camera Control Bus we need to connect our camera to Arduino.

3.1.1 Servo Motor

We would use SG90 servo motor in order to move our camera. We chose this model simply due to its ease of use and its affordable cost.

Figure 3: SG90 servo

3.1.2 ESP 8266 Wifi Module

We have already bought this module for our lab. This module is used for sending and receiving UDP packets from server.

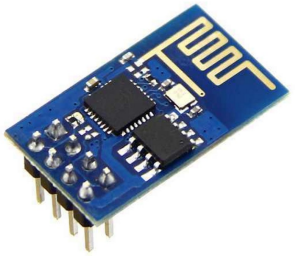


Figure 4: ESP 8266 Wifi Module

```
 /------------------------------\
 |              chip      1   8 |
 | Ant                    2   7 |
 | enna       processor   3   6 |
 |                        4   5 |
 \------------------------------/
ESP8266    TM4C123
  1 URxD     PB1   UART out of TM4C123, 115200 baud
  2 GPIO0          +3.3V for normal operation (ground to flash)
  3 GPIO2          +3.3V
  4 GND      Gnd   GND (70mA)
  5 UTxD     PB0   UART out of ESP8266, 115200 baud
  6 Ch_PD          chip select, 10k resistor to 3.3V
  7 Reset    PB5   TM4C123 can issue output low to cause hardware reset
  8 Vcc            regulated 3.3V supply with at least 70mA
```

Figure 5: PIN connections of ESP 8266

3.1.3 Camera

For camera we intend to use OV7670. This camera is connected to Arduino which captures the images for java-server..

3.2 Software Layer

There will be three parts on our software layer. Our server gets the image and analyzes it then sends UDP packet, our launchpad receives UDP packet and moves servo motors accordingly.

Figure 6: OV7670

```
OV7670 PIN connections:

VSYNC - PIN2
XCLCK - PIN3 (must be level shifted from 5V -> 3.3V)
PCLCK - PIN12
SIOD - A4 (I2C data) - 10K resistor to 3.3V
SIOC - A5 (I2C clock) - 10K resistor to 3.3V
D0..D3 - A0..A3 (pixel data bits 0..3)
D4..D7 - PIN4..PIN7 (pixel data bits 4..7)
3.3V - 3.3V
RESET - 3.3V
GND - GND
PWDN - GND
```

Figure 7: PIN connections of OV7670

3.2.1 Getting image from camera and Analyzing the frame

Since we are limited with our launchpad, so we used Arduino UNO for capturing images. Arduino sends the frames to computer 115200 baudrate. Then server would create a UDP packet that contains only red value of rgb then sends it to the launchpad.

```
void startNewFrame(uint8_t pixelFormat) {
  UDR0 = 0x00;
  pixelSendingDelay();
  UDR0 = COMMAND_NEW_FRAME;
  pixelSendingDelay();

  // frame width
  UDR0 = (lineLength >> 8) & 0xFF;
  pixelSendingDelay();
  UDR0 = lineLength & 0xFF;
  pixelSendingDelay();

  // frame height
  UDR0 = (lineCount >> 8) & 0xFF;
  pixelSendingDelay();
  UDR0 = lineCount & 0xFF;
  pixelSendingDelay();

  // pixel format
  UDR0 = (pixelFormat);
  pixelSendingDelay();
}
```

Figure 8: Start new frame

Frames are drew with this function;

```
// update image in a separate thread so it would not block reading data
new Thread(() -> {
  synchronized (imageContainer) {
    Graphics2D g = imageBuffer.createGraphics();
    int fromLine = lineIndex != null ? lineIndex : 0;
    int toLine = lineIndex != null ? lineIndex : frame.getLineCount() - 1;

    for (int y = fromLine; y <= toLine; y++) {
        String temp = "";
        for (int x = 0; x < frame.getLineLength(); x++) {
            if (x < MAX_IMAGE_W && y < MAX_IMAGE_H) {
                g.setColor(frame.getPixelColor(x, y)); // gets r, g, b values of the pixel
                g.drawLine(x, y, x, y);
            }
        }
    }
    imageContainer.repaint();
  }
}).start();
```

Figure 9: Draw Frame Function

This is the content of UDP packet which is prepared by Java-server;

```
new Thread(() -> {

    for(int x=0; x<160;x++) {
        int red = frame.getPixelColor(x,lineIndex).getRed();
        int green = frame.getPixelColor(x,lineIndex).getGreen();
        int blue = frame.getPixelColor(x,lineIndex).getBlue();
        rArray[x] += isRed(red, green, blue);
    }

    if(lineIndex == 119) {
        for(int j=0;j<160;j++) {
            int index = j / 16;
            sArray[index] += rArray[j];
        }

        String result = "";

        for(int i=0;i<10;i++) {
            String temp = String.format("%04d" , sArray[i]);
            result += temp + "-";
        }

        System.out.println(result);
        tempResult = result;

        sArray = new int[10];
        rArray = new int[160];
    }
}).start();
```

Figure 10: UDP Packet

Each packet sent by this function;

```
public void sendUDPtoMCU(String row) {
    DatagramPacket packet = new DatagramPacket(row.getBytes(), row.length(), this.address, this.port);
    try {
        socket.send(packet);
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

Figure 11: UDP send function

3.2.2 Receiving UDP packet from Wifi Module

Our launchpad would receive UDP packets that holds the information which is later used for moving servo motors to corresponding degree. For receiving UDP packets we will use ESP8266 wifi module that is capable of such things. Usage of ESP8266 wifi module;

```
printf("\n\r-----------\n\rSystem starting...\n\r");
ESP8266_Init(115200);      // connect to access point, set up as client
ESP8266_GetVersionNumber();
EnableInterrupts();
ESP8266_GetStatus();
  if(ESP8266_MakeTCPConnection("192.168.43.245")){ // open socket in server
    ESP8266_SendTCP(Fetch);
}
printf("\n\rUDP sent!\n\r");
```

Figure 12: ESP8266 wifi module usage

3.2.3 Moving servo motors accordingly

In order to move servo, our microcontroller would analyze data of new coming UDP packets and decide which direction servo motor should turn. Then by sending proper commands to servo motor, it allows camera to track the color. Then decision was made on Systick Handler for each frame.

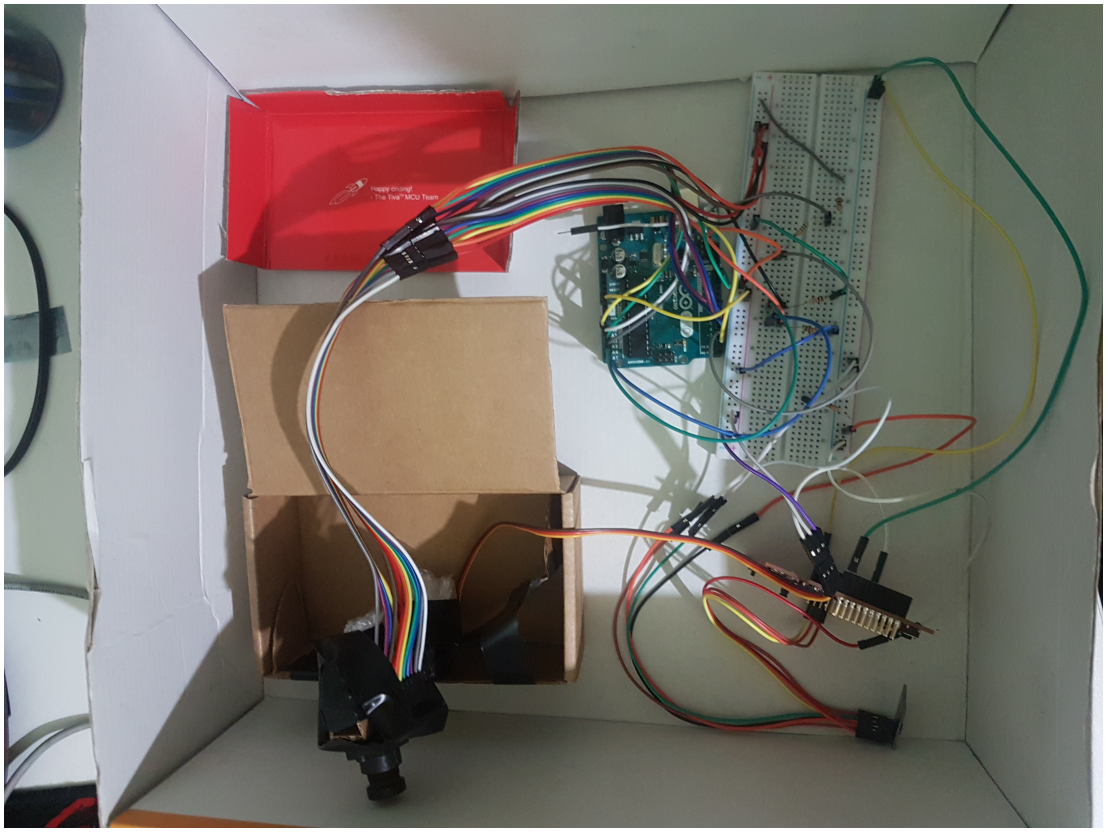
```
void SysTick_Handler(void){
  fiveSec++;
  if(fiveSec % 5 == 0){
    fiveSec = 0;
    ESP8266_SendTCP(Fetch);

    LED_BlueToggle();
    int i, j;
    char str[52];
    for(i=0;i<52;i++){
      char c = returnBuffer(i);
      str[i] = c;
      printf("%c", c);
    }
    decisionArray[0] = 0;
    decisionArray[1] = 0;
    char* token = strtok(str, "-");
```

Figure 13: First part of Systick Handler

```
  for(j=0;j<10;j++){
    int temp = atoi(token);
    if(j<5){
      decisionArray[0] += temp;
    }else{
      decisionArray[1] += temp;
    }
    token = strtok(NULL, "-");
  }
  if(decisionArray[0] > 500 || decisionArray[1] > 500){
    if(decisionArray[0] > decisionArray[1] + (decisionArray[0]/10)){
      if(currentServoAngle < 180){
        currentServoAngle += 30;
        printf("\n Servo is turning left!");
      }
    }else if(decisionArray[1] > decisionArray[0] + (decisionArray[1]/10)){
      if(currentServoAngle > 0){
        currentServoAngle -= 30;
        printf("\n Servo is turning right!");
      }
    }
  }
  SetServoAngle(currentServoAngle);
}
```

Figure 14: Second part of Systick Handler

4 Photo Documentation



5 Timeline

The project spans 4-5 weeks. From first and second week, we implemented our server and in third week we implemented the UDP communication between server and the launchpad. Finally for the last weeks we implemented moving servo mechanism and we need to optimize our code, fix bugs and then test the device.

6 Responsibilities

6.0.1 Çağdaş Utku Tonbul

Main focus on software side of the project
To implement server side of project
To implement image capturing
To implement communication between server and launchpad

6.0.2 Furkan Karakökçek

Main focus on hardware side of the project
To implement connections of our external hardware
To implement server side of project
Programming servo motors to move according to information

7 Budget

- Servo Motor(20TL)
- External Camera(40TL)
- Arduino Uno R3(140TL)