Vietnamese-German University

PROJECT 2

Object Tracking using Cosine Similarity

Name: Phan Tâm Như

ID: 10421122

09/11/2022

I. Aim

This project is about object tracking by using cosine similarity and C implementation to run through a set of pictures and find the similarity between the given data and the source template.

II. Object Tracking

Object tracking is an application of deep learning where the program tracks the movement of an object, through the process, the computers are able to detect, understand and keep an eye on the objects across the still images or video.

III. Procedure

1. Directory structure

To read and write the images in C, there are two specific headers to be imported:

- The library to read an image:

https://github.com/nothings/stb/blob/master/stb_image.h

- The library to write an image:

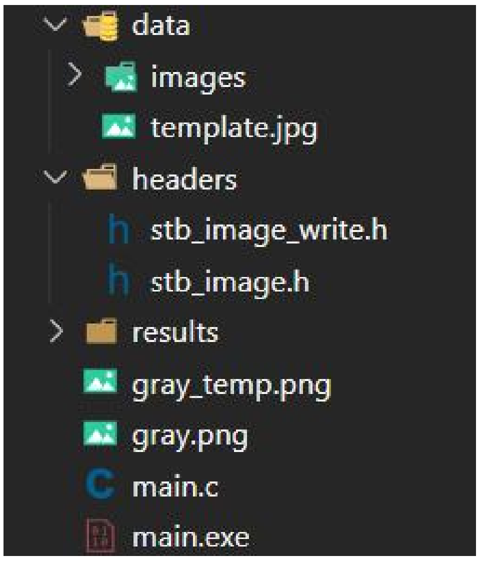
https://github.com/nothings/stb/blob/master/stb_image_write.h

The project is based on a given set of data by the instructors, which included 63 images and a template for comparison:

https://www.dropbox.com/sh/6ptn2fhnjr3glpy/AACyRsWKdXuQii8IB1tYtewta?dl=0

The images are put in a file called data and the outputs are put in a file called results.

This is a picture of the project's directory structure:



```
∨ data
  > images
    template.jpg
∨ headers
    stb_image_write.h
    stb_image.h
> results
  gray_temp.png
  gray.png
  main.c
  main.exe
```

2. Convert the pictures to grayscale

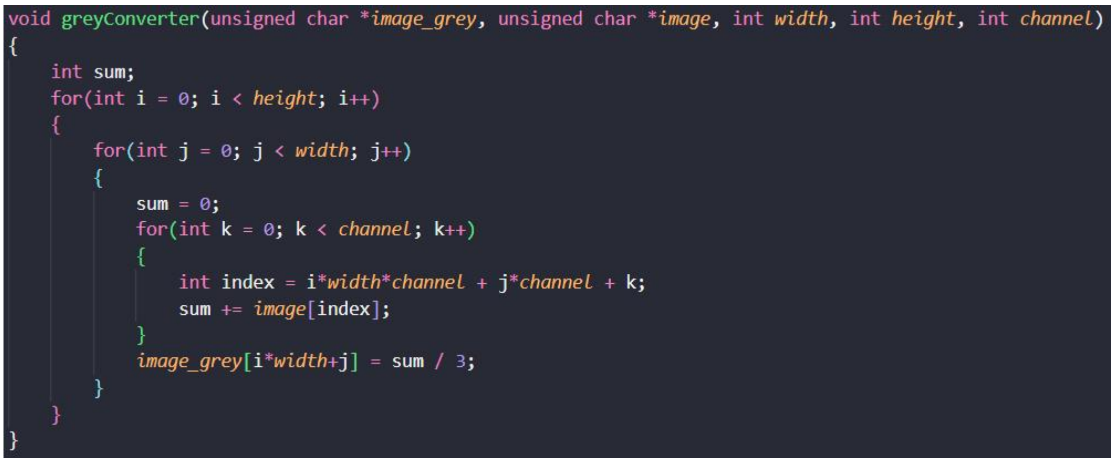
A picture has 3 channels of color called RGB, and it is stored as a data array that defines red, green, and blue color components for each individual pixel.

The reason for this step is because the RGB image contains a lot of information which may not required for the processing, and by doing this the complexity and the running time of the program can be reduced.

Applying the Average method below, the template and the image are changed into grayscale:

$$\boxed{\text{Grayscale} = (R + G + B\,)\,/\,3}$$

Below is the function of converting into grayscale:



```
void greyConverter(unsigned char *image_grey, unsigned char *image, int width, int height, int channel)
{
    int sum;
    for(int i = 0; i < height; i++)
    {
        for(int j = 0; j < width; j++)
        {
            sum = 0;
            for(int k = 0; k < channel; k++)
            {
                int index = i*width*channel + j*channel + k;
                sum += image[index];
            }
            image_grey[i*width+j] = sum / 3;
        }
    }
}
```

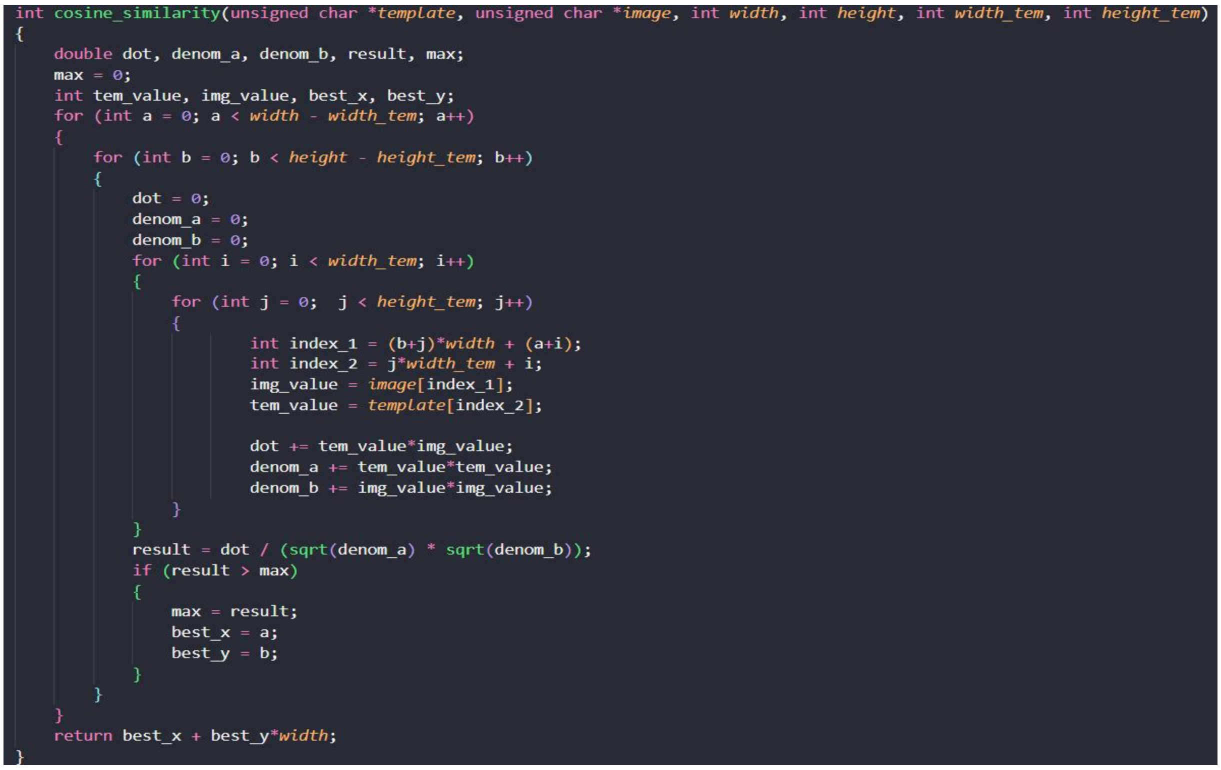
3. Cosine Similarity

This project is based on the algorithm of cosine similarity - a type of distance measure.

$$\text{cosine similarity} = S_C(A,B) := \cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2}\sqrt{\sum\limits_{i=1}^{n} B_i^2}},$$

I wanted to get the maximum result in order to find the domain that are exactly the same with the original template.

Below is the function of cosine similarity:



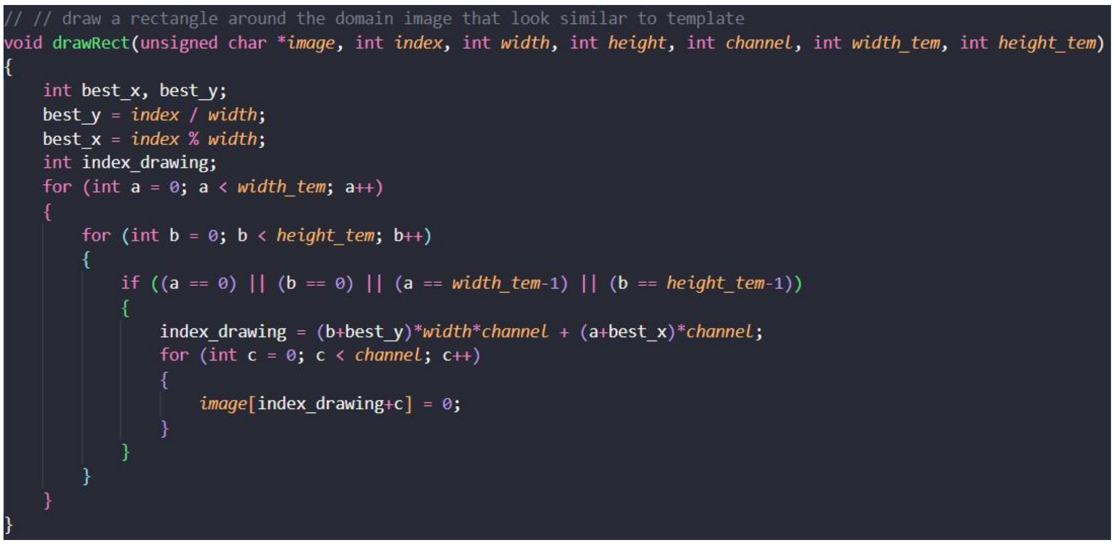
```
int cosine_similarity(unsigned char *template, unsigned char *image, int width, int height, int width_tem, int height_tem)
{
    double dot, denom_a, denom_b, result, max;
    max = 0;
    int tem_value, img_value, best_x, best_y;
    for (int a = 0; a < width - width_tem; a++)
    {
        for (int b = 0; b < height - height_tem; b++)
        {
            dot = 0;
            denom_a = 0;
            denom_b = 0;
            for (int i = 0; i < width_tem; i++)
            {
                for (int j = 0;  j < height_tem; j++)
                {
                        int index_1 = (b+j)*width + (a+i);
                        int index_2 = j*width_tem + i;
                        img_value = image[index_1];
                        tem_value = template[index_2];

                        dot += tem_value*img_value;
                        denom_a += tem_value*tem_value;
                        denom_b += img_value*img_value;
                }
            }
            result = dot / (sqrt(denom_a) * sqrt(denom_b));
            if (result > max)
            {
                max = result;
                best_x = a;
                best_y = b;
            }
        }
    }
    return best_x + best_y*width;
}
```

4. Draw a rectangle

To detect the object, I write a function to draw a rectangle around the domain image that look similar to the template.

After getting the coordinate of the domain image (best_x, best_y), I convert the pixels around it into black and get the following result.

Below is the function of drawing a rectangle:

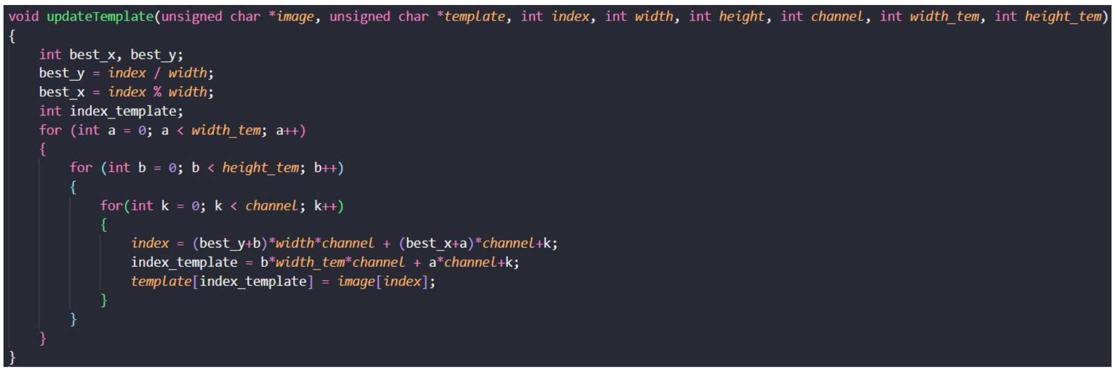


```
// // draw a rectangle around the domain image that look similar to template
void drawRect(unsigned char *image, int index, int width, int height, int channel, int width_tem, int height_tem)
{
    int best_x, best_y;
    best_y = index / width;
    best_x = index % width;
    int index_drawing;
    for (int a = 0; a < width_tem; a++)
    {
        for (int b = 0; b < height_tem; b++)
        {
            if ((a == 0) || (b == 0) || (a == width_tem-1) || (b == height_tem-1))
            {
                index_drawing = (b+best_y)*width*channel + (a+best_x)*channel;
                for (int c = 0; c < channel; c++)
                {
                    image[index_drawing+c] = 0;
                }
            }
        }
    }
}
```

5. Update the template

This function is needed for this project since in the set of data, there are some pictures that the object doesn’t in the central angle. By doing this step, the template will be updated to make sure that the object is always being followed and detected.

Below is the function of updating the template:

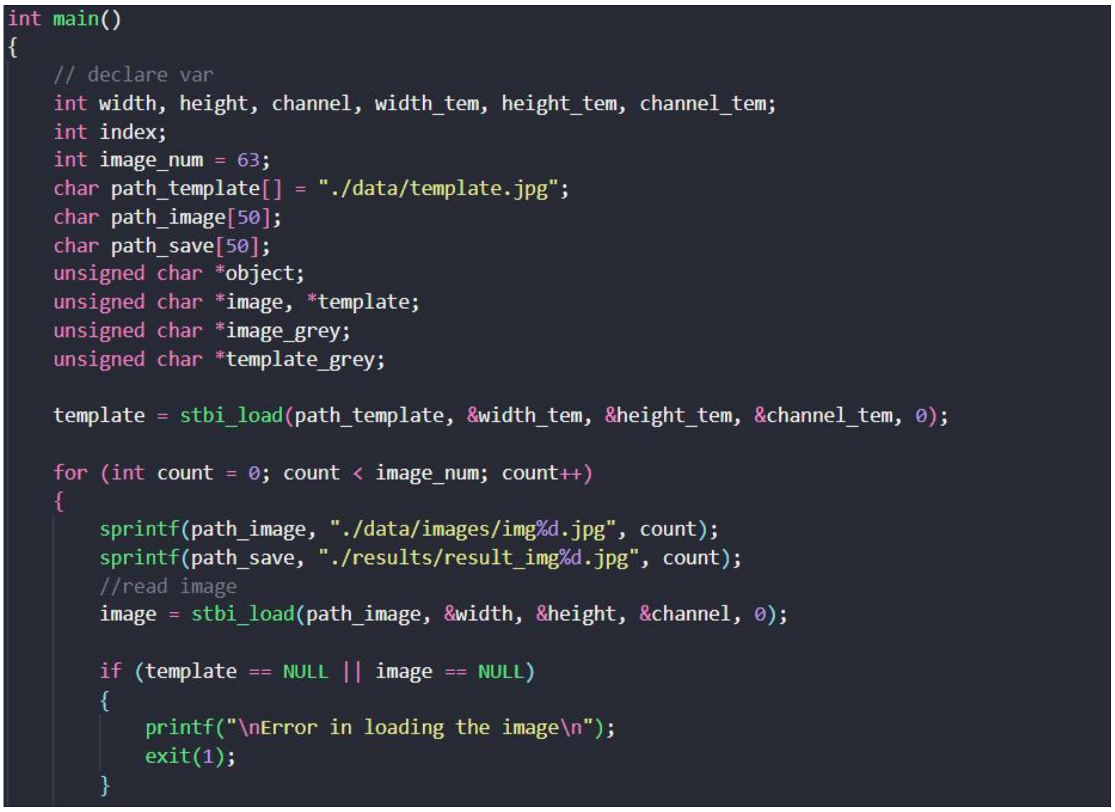


```
void updateTemplate(unsigned char *image, unsigned char *template, int index, int width, int height, int channel, int width_tem, int height_tem)
{
    int best_x, best_y;
    best_y = index / width;
    best_x = index % width;
    int index_template;
    for (int a = 0; a < width_tem; a++)
    {
        for (int b = 0; b < height_tem; b++)
        {
            for(int k = 0; k < channel; k++)
            {
                index = (best_y+b)*width*channel + (best_x+a)*channel+k;
                index_template = b*width_tem*channel + a*channel+k;
                template[index_template] = image[index];
            }
        }
    }
}
```

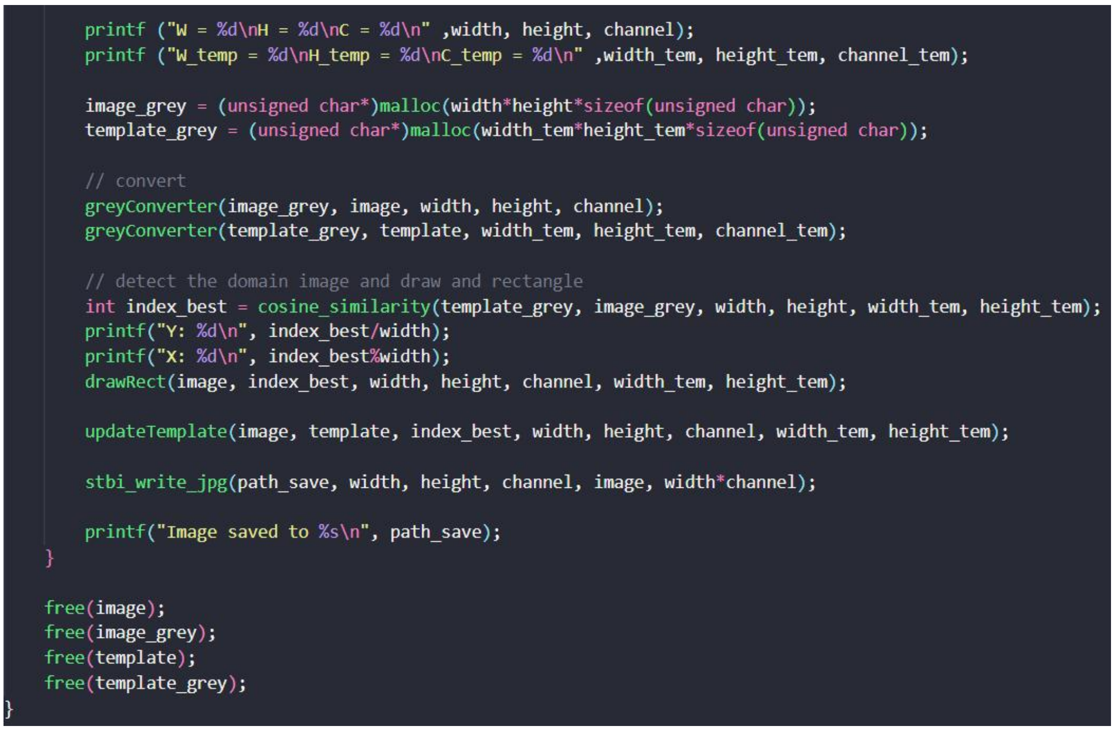
6. Main function

The main function is all about declaring the variables that I used in the program, since there are 63 images that needed to detect, there is a variable to count and then loop the above functions throughout the process.

Below is the main function:



```
int main()
{
    // declare var
    int width, height, channel, width_tem, height_tem, channel_tem;
    int index;
    int image_num = 63;
    char path_template[] = "./data/template.jpg";
    char path_image[50];
    char path_save[50];
    unsigned char *object;
    unsigned char *image, *template;
    unsigned char *image_grey;
    unsigned char *template_grey;

    template = stbi_load(path_template, &width_tem, &height_tem, &channel_tem, 0);

    for (int count = 0; count < image_num; count++)
    {
        sprintf(path_image, "./data/images/img%d.jpg", count);
        sprintf(path_save, "./results/result_img%d.jpg", count);
        //read image
        image = stbi_load(path_image, &width, &height, &channel, 0);

        if (template == NULL || image == NULL)
        {
            printf("\nError in loading the image\n");
            exit(1);
        }
```

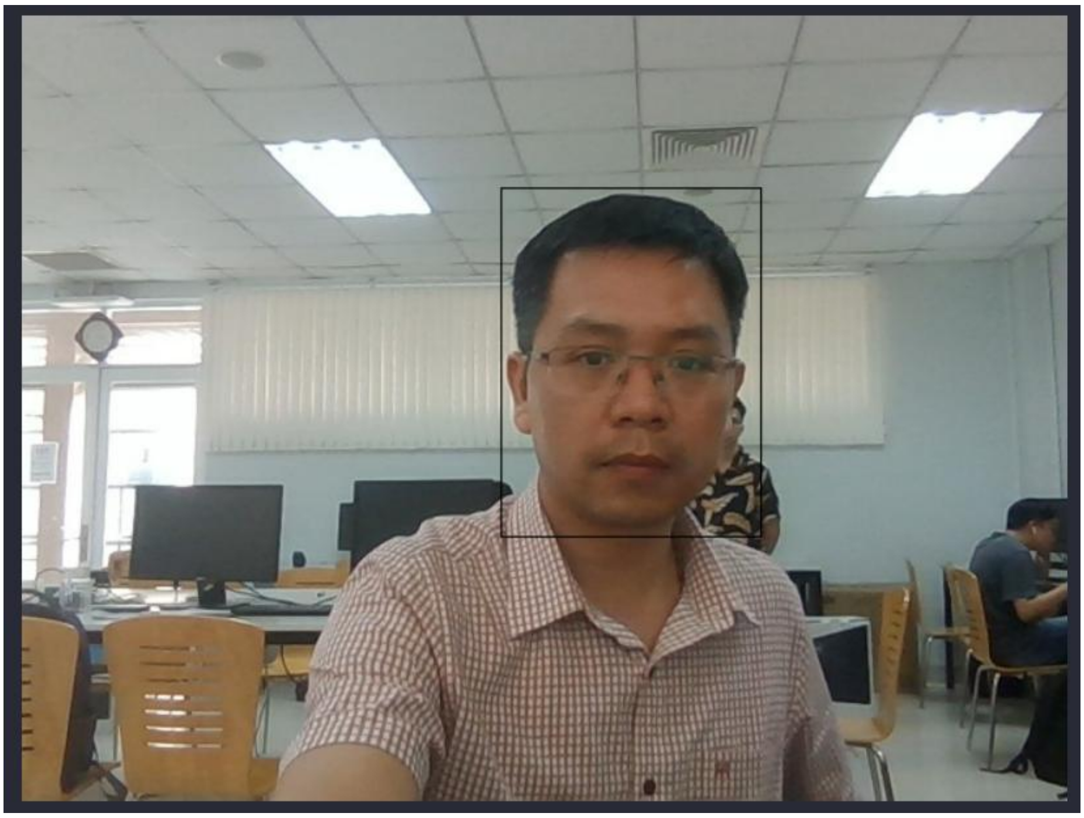


```c
        printf ("W = %d\nH = %d\nC = %d\n" ,width, height, channel);
        printf ("W_temp = %d\nH_temp = %d\nC_temp = %d\n" ,width_tem, height_tem, channel_tem);

        image_grey = (unsigned char*)malloc(width*height*sizeof(unsigned char));
        template_grey = (unsigned char*)malloc(width_tem*height_tem*sizeof(unsigned char));

        // convert
        greyConverter(image_grey, image, width, height, channel);
        greyConverter(template_grey, template, width_tem, height_tem, channel_tem);

        // detect the domain image and draw and rectangle
        int index_best = cosine_similarity(template_grey, image_grey, width, height, width_tem, height_tem);
        printf("Y: %d\n", index_best/width);
        printf("X: %d\n", index_best%width);
        drawRect(image, index_best, width, height, channel, width_tem, height_tem);

        updateTemplate(image, template, index_best, width, height, channel, width_tem, height_tem);

        stbi_write_jpg(path_save, width, height, channel, image, width*channel);

        printf("Image saved to %s\n", path_save);
    }

    free(image);
    free(image_grey);
    free(template);
    free(template_grey);
}
```

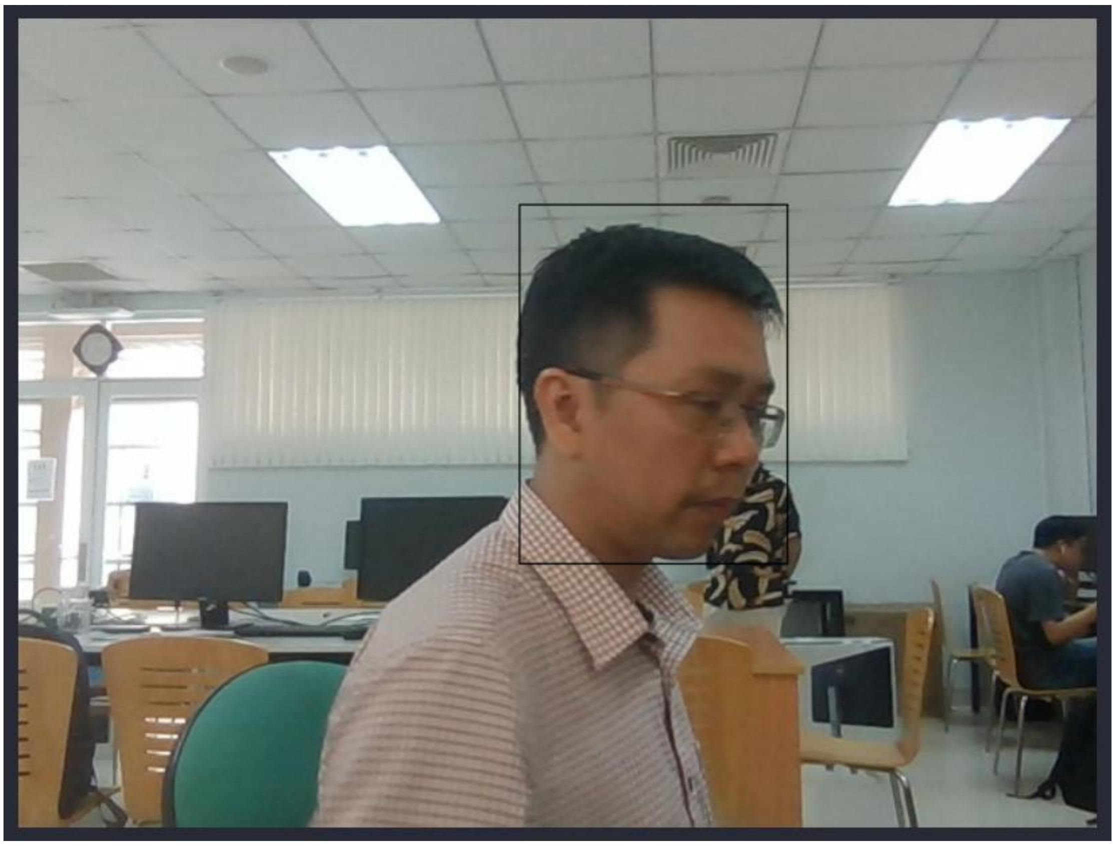
IV. Outcome

The running time in overall is quite long since there are many loops to go through. Below are some of the outcomes that I have:

+ Image 0:



+ Image 20:



+ Image 40: