

Project 2 – Object Tracking

Using cosine similarity algorithm

Huỳnh Lê An Phú | Programming 1 | 07/11/2022

Instructor: Prof. Đinh Quang Vinh

# Project Aim and Project Structure

## Project aim:

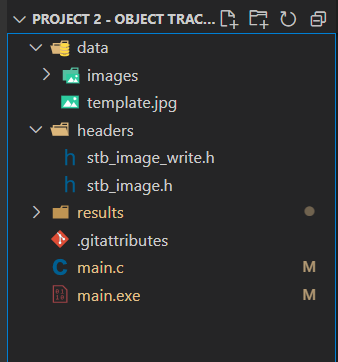
This project aims to create a program to track a specific given object in a series of images. The algorithm in the code is called Cosine Similarity, which considers the pixel and channel of an image (or a part of it) as a vector, and calculate the different of the angle between that vector and the vector of the given object.

More details will be presented in the next Chapter: Cosine Similarity.

## project structure

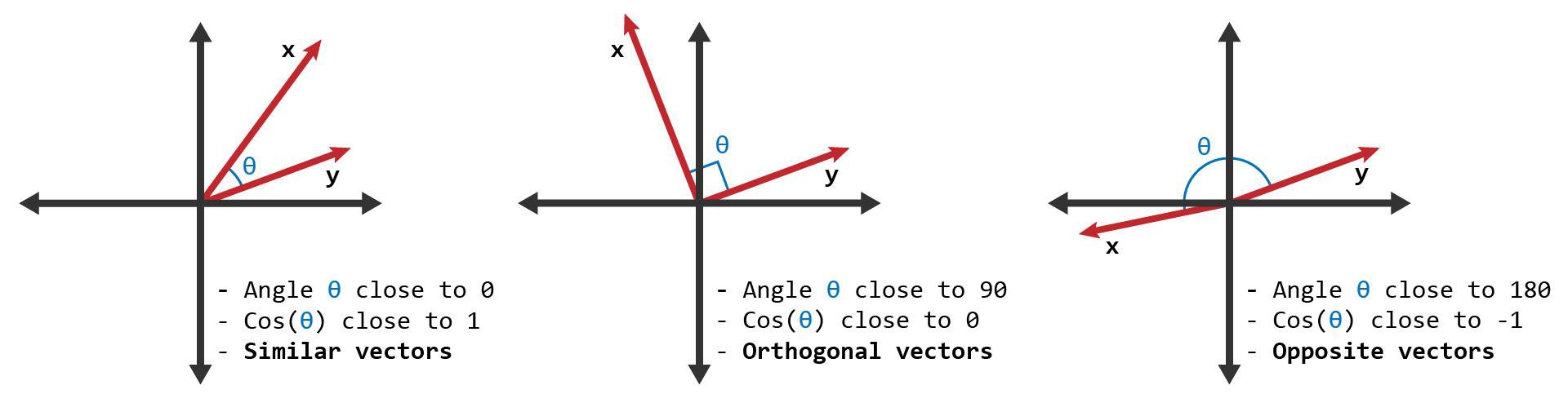
For the sake of this project, I will try to track the face of my professor in 63 images. The object (the face) needed to be tracked is stored in a file “template.jpg”

The project is uploaded to Github via this link: *https://github.com/phuhuynhlean/Proj.2-ObjectTracking*

1. Folder *data* is used to store processing images (in subfolder *images*) and object image. In total: 63 images for processing and 1 template.
2. Folder *headers* is used to store header files (external library in C). The headers can be found online and are used to get and write images.
3. Folder *results* is the directory for storing processed images (image with the object bordered)
4. All the code for the project is written inside *main.c* in C language and built into *main.exe*

# Some of the mechanisms and logics behind the program

## Cosine Similarity



First, let’s take 2-dimensional vectors as an example. Cosine similarity measures the cosine of the angle between two vectors.

The closer the vectors are, the closer the cosine of that angle is to 1 (cosine of a zero-degree angle is 1).

This property still applies to n-dimensional vectors. Similar vectors have the cosine similarity closer to 1.

The Formula for calculating cosine similarity of 2 vectors A and B of n dimension.

## Multidimensional Arrays in C / C++ - GeeksforGeeksImage cosine similarity

Images themselves are just series of values stored in a very long array. Therefore, they can be seen as vectors themselves.

Using cosine similarity, we can measure the similarity between the images.

This method has its advantages and disadvantages, which shall be discussed at the end of the report.

## Grayscale an image

### Why do we need to grayscale?

Normally, a pixel in a colored image has 3 channels storing the Red, Green, Blue (RGB) values of that pixel. Converting a color image to a grey image will reduce the number of value needed to be processing to a third.

Grayscale may decrease the accuracy of the image but I believe in this project, this is worthy as keeping the original color image will triple the processing time.

### How to grayscale?

There is not only one method to grayscale an image. Some of the typical one includes:

### Taking the average value of RGB

### Multiplying the value of RGB with certain weights

### Multiplying the value of RGB with its percentage

In this project, I choose the third method to imply.

The percentage of each one is calculated by dividing its value with the sum of 3 channel values.

For example, I have the value of 3 channels in a pixel as below:

Red:  
170

Blue:  
120

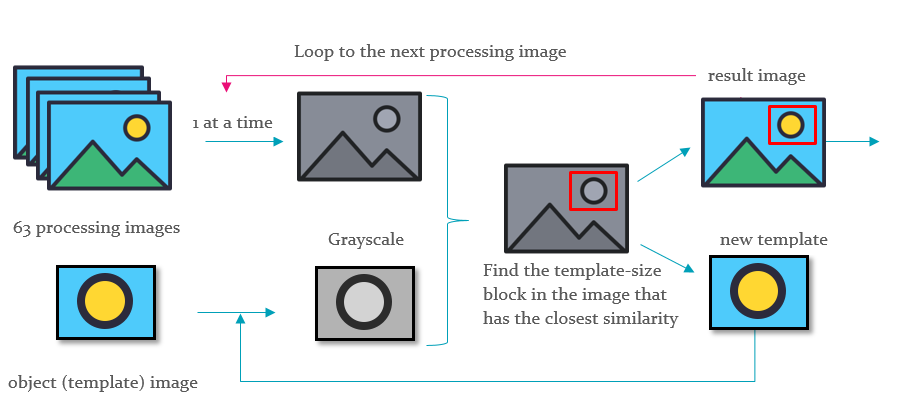
Green:  
150

Greyscale  
(170^2 + 150^2 + 120^2)/(170+150+120)

# Code Execution

## code explanation:

Here is a diagram of what happened in the program.



As depicted above, there are 4 main steps involved in the program

1. Load the processing image and template to the program using the external library (available online)
2. Grayscale the processing image and the template
3. Calculate the similarity of each template-size block in the processing image and pick out the index of the block that has the highest cosine similarity (best block).
4. Draw a rectangle outline bordering the best block and write image to results folder. Update the template based on that block. Repeat the first step with the next image until finished.

The specific code for all the functions above will be presented in the next 4 pages of the report.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Step: | 1st Step: Loading | 2nd Step: Grayscale | 3rd Step: cosine similarity | 4th Step: |
| Where: | Library declare  Main function | bwConverter() | objectDetector() | drawRectandUpdateTemplate() |

## Full code:

#include <time.h>

#include <stdio.h>

#include <math.h>

#define STB\_IMAGE\_IMPLEMENTATION

#include "./headers/stb\_image.h"

#define STB\_IMAGE\_WRITE\_IMPLEMENTATION

#include "./headers/stb\_image\_write.h"

float object\_sqrt(unsigned char \*object, int object\_width, int object\_height) {

    float sum = 0;

    for (int i = 0; i < object\_width \* object\_height; i++) {

        sum += object[i] \* object[i];

    }

    return sqrt(sum);

}

void bwConverter(unsigned char \*input, unsigned char \*output, int width, int height, int channel) {

    for (int i = 0; i < width ; i++) {

        for (int j=0; j< height; j++) {

            int index = j \* width \* channel + i \* channel;

            int index\_out = j \* width + i;

            unsigned char red = input[index];

            unsigned char green =input[index + 1];

            unsigned char blue = input[index + 2];

            output[index\_out] = (red\*red + green\*green+blue\*blue+1)/(red+green+blue+1);

        }

    }

}

int objectDetector(unsigned char \*image, int width, int height, unsigned char \*object, int object\_width, int object\_height) {

    float max, similarity;

    int best\_x, best\_y,img\_element, obj\_element;

    float sum, sqrt1, sqrt2;

    similarity = 0;

    max = 0.1;

    sqrt2 = object\_sqrt(object, object\_width, object\_height);

    for (int i = 0; i < width - object\_width; i++) {

        for (int a = 0; a < object\_width ; a++){

            for (int b = 0; b < object\_height; b++){

                img\_element = image[(b) \* width + (a+i)];

                obj\_element = object[b \* object\_width + a];

                sqrt1 += img\_element \* img\_element;

            }

        }

        for (int j = 0; j < height - object\_height; j++) {

            sum = 0;

            if (j!=0) {

            }

            for (int a = 0; a < object\_width ; a++){

                for (int b = 0; b < object\_height; b++){

                    img\_element = image[(b+j) \* width + (a+i)];

                    obj\_element = object[b \* object\_width + a];

                    sum += img\_element \* obj\_element;

                }

            }

            similarity = sum / (sqrt(sqrt1) \* sqrt2);

            if (similarity > max) {

                max = similarity;

                best\_x = i;

                best\_y  = j;

            }

        }

    }

    return best\_y \* width + best\_x;

}

void drawRectandUpdateTemplate(unsigned char \*image, int width, int height, int channel, int new\_index, int object\_width, int object\_height, unsigned char\* object) {

    int best\_y, best\_x;

    best\_y = new\_index / (width);

    best\_x = new\_index % (width);

    printf("Best x: %d, Best y: %d\n", best\_x, best\_y);

    int index, index\_object, i, j;

    for (i = 0; i<object\_height; i++){

        for (j = 0; j<object\_width; j++){

            for (int k = 0; k<3; k++){

                index = (best\_y + i) \* width \* channel + (best\_x +  j) \* channel + k;

                index\_object =  i \* object\_width \* channel + j \* channel + k;

                object[index\_object] = image[index];

            }

        }

    }

    for (i = 0; i <object\_width; i++) {

        for (int k = 0; k< 3; k++){

            index = (best\_y ) \* width \* channel + (best\_x+i) \* channel + k;

            image[index] = 0;

            index = index  + (object\_height-1) \* width \* channel;

            image[index] = 0;

        }

    }

    for (i = 1; i <object\_height -1; i++) {

        for (int k = 0; k< 3; k++){

            index = (best\_y + i) \* width \* channel + (best\_x) \* channel + k;

            image[index] = 0;

            index = index + (object\_width-1) \* channel + k;

            image[index] = 0;

        }

    }

}

int main() {

    clock\_t begin = clock();

    int width, height, channel;

    int width2, height2, channel2;

    char image\_path[50];

    char object\_path[50] = "./data/template.jpg";

    char save\_path[50];

    unsigned char \*image\_grey, \*object\_grey;

    unsigned char \*image, \*object;

    for (int count = 0; count < 63; count++){

        clock\_t tic = clock();

        sprintf(image\_path, "./data/images/img%d.jpg", count);

        sprintf(save\_path, "./results/result\_%d.jpg", count);

        image = stbi\_load(image\_path, &width, &height, &channel, 0);

        if(image == NULL) {

            printf("Error: image not found!\n");

            exit(1);

        }

        if (count == 0){

          object = stbi\_load(object\_path, &width2, &height2, &channel2, 0);

          if(object == NULL) {

              printf("Error: object not found!\n");

              exit(1);

          }

        }

        image\_grey = (unsigned char \*)calloc(width\*height, sizeof(unsigned char));

        bwConverter(image, image\_grey, width, height, channel);

        object\_grey = (unsigned char \*)calloc(width2\*height2, sizeof(unsigned char));

        bwConverter(object, object\_grey, 160, 214, 3);

        // //detect object

        printf("Detecting object...\n");

        int index = objectDetector(image\_grey, width, height, object\_grey, width2, height2);

        drawRectandUpdateTemplate(image, width, height, channel, index, width2, height2, object);

        //save image

        stbi\_write\_jpg(save\_path, width, height, channel, image, width \* channel);

        printf("New image successfully saved to %s\n", save\_path);

        clock\_t toc = clock();

        double time = (double)(toc - tic) / CLOCKS\_PER\_SEC;

        printf("Processing time %f - image: %d\n", time, count);

    }

    free(image);

    free(object);

    free(image\_grey);

    free(object\_grey);

    clock\_t end = clock();

    double time\_spent = (double)(end - begin) / CLOCKS\_PER\_SEC;

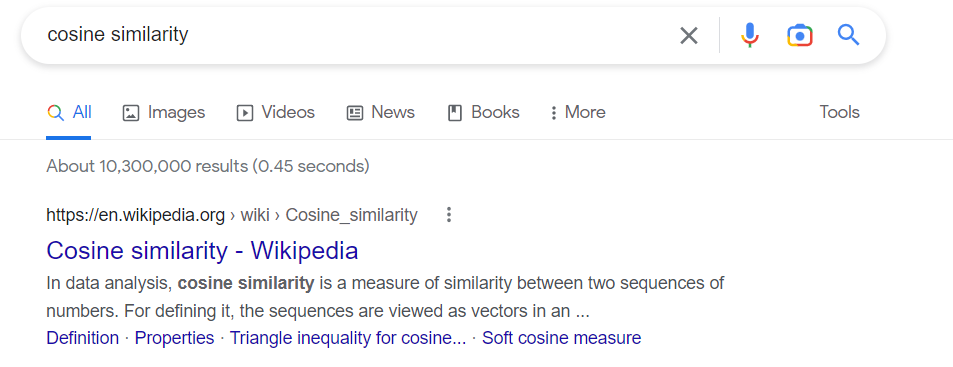
    printf("Processing time: %f", time\_spent);

    return 0;

}

# Advantages and Disadvantages:

This algorithm is easy to use and quite popular. Therefore, source code and reference for it is common on multiple forums and medias.



On the other hand, implementing the Cosine Similarity algorithm without cleaning the image (for example, processing background) can cause disturbance in the tracking function, as also encountered in this project:



In addition, the object itself should not change in its shape, size, or appearance very fast since those aspects can severely affect the effectiveness of the algorithm.