Project 2 Report - C Programming

Do Minh Quang

November 2022

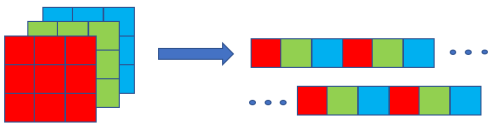
***Abstract*—This report will propose an object detection simulation in C programming. Overall, it consists of 3 main parts: fundamental knowledge for this project, an outline of steps and a detailed explain in each step.**

I. Fundamental Knowledge

The components required for this project includes:

- Image interaction (Load and save)
  Provided with the header files `stb_image.h` and `stb_image_write.h`. They include functions:
  - `stbi_load()` for loading images and return a char pointer to an image.
  - `stbi_write_png()` for saving an image from its char pointer.

  Images are represented as 1-D array in C (its color value by pixel and by channel is placed consecutively instead of 3 dimensional array like in Python. This will affect how we iterate through the image.)



- Image color conversion (RGB to Greyscale)
  Since the detection requires the object in the image and the template to look the same, it is unnecessary to match the color completely. This implies that the template and background images can be converted into greyscale but still keep the object intact for comparison. In addition, this conversion also helps reducing computing time.

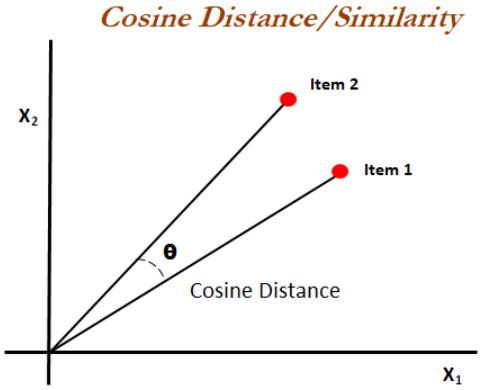
  In this project, we convert RGB to Greyscale according to the formular:

$$x = 0.299r + 0.587g + 0.114b \tag{1}$$

  This is because the colors are not weighted equally. Since pure green is lighter than pure red and pure blue, it has a higher weight. Pure blue is the darkest of the three, so it receives the least weight.[1]

- Cosine similarity (Metric for comparison)

$$\cos(\theta) = \frac{A \cdot B}{\|A\| \, \|B\|} \tag{2}$$



  This metric generates a result range from -1 to 1. On geometry view, it measures the gap distance between 2 vectors: 1 means they have same direction and -1 means they are opposite. This implies that the nearer the value to 1, the 2 items are more closely related. This will be used to compare the template with different area of the background image.

II. General Idea

Before jumping into the coding part, a plan is needed to be made. From that point, the coding part can follow up and less likely to encounter errors.

These are the basic steps for this project:

1) Load template and every single background image.
2) Iterate the template over the background row by row and column by column.
3) Compute the Cosine similarity between the template and the area which it is covering.
4) Find the area that has the highest Cosine similarity value, there draw red border and update the template by that area.
5) Repeat for the remaining images.

III. Actual Code

You can view the below links to examine the source code and live demo of this project!
Source code: Click to see code
Demo video: Click to see video

The program has 2 functions: `main` and `process_image`. The `main` function is mandatory and `process_image` will scan through 62 images with the template and detect the object on those images.

```
unsigned char processing image

int main() {
```

The `main` function will load the template and run the `process_image` function.

Step 1: Create a for loop from 0 to 62. In each loop, create the path for loading and saving the images.

```
for (int o = 0; o <= 62; o++) {

    char temp[5];

    sprintf(temp, "%d", o);

    char path_background[] = "./images/img";

    char jpg[] = ".jpg";

    strcat(path_background, temp);
    strcat(path_background, jpg);
    printf("Load image %s \n", path_background);
```

Then, preprocess images by converting the template and background from RGB to greyscale.

```
char * tem = (char * ) malloc(width * height);

for (int i = 0; i < height; i++) {
    for (int j = 0; j < width; j++) {
        tem[i * width + j] = (0.299 * Template[i * width * ch
    }
}

char * back = (char * ) malloc(width1 * height1);

for (int i = 0; i < height1; i++) {
    for (int j = 0; j < width1; j++) {
        back[i * width1 + j] = (0.299 * background[i * width1
    }
}
```

Step 2: Iterate through pixel by pixel in each of the background image. By increasing the step of sliding the template, the process will become faster but might cause error. In my project, I slide the template by 20 pixel each turn.

```
for (int i = 0; i < height1 - height + 1; i = i + 20) {

    for (int j = 0; j < width1 - width + 1; j = j + 20) {
```

Step 3: Calculate the cosine similarity of each turn between the template and the area of the background image.

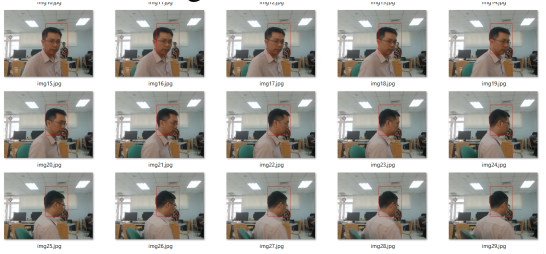
```
for (int i = 0; i < height1 - height + 1; i = i + 20) {

    for (int j = 0; j < width1 - width + 1; j = j + 20) {

        double cosine = 0;
        double dot = 0;
        double A = 0;
        double B = 0;

        for (int h = 0; h < height; h++) {
            for (int w = 0; w < width; w++) {

                A += pow(tem[h * width + w], 2.0);
                B += pow(back[(i + h) * width1 + (j + w)], 2.0);
                dot += tem[h * width + w] * back[(i + h) * width1 + (j + w)];

            }
        }
        cosine += dot / (pow(A, 0.5) * pow(B, 0.5));
        if (cosine > max) {
            max = cosine;
            x = j;
            y = i;
        }
        if (i * j % 20000 == 0 && i * j != 0) {

            printf("%c", 219);
        }

    }
}
```

Step 4: After calculation, we can determine the best area fitting the template. Then base on the cordinate `X` and `Y`, we draw a red border to capture that area. Note: here I set the pixel to (255,0,0) equals Red and the thickness of the border is 2 pixel.

```
if (cosine > max) {
    max = cosine;
    x = j;
    y = i;
}
```

```
for (int w = 0; w < width; w++) {
    background[(y - 1) * width1 * channel + (x + w) * channel + 0] = 255;
    background[(y - 1) * width1 * channel + (x + w) * channel + 1] = 0;
    background[(y - 1) * width1 * channel + (x + w) * channel + 2] = 0;

    background[(y - 2) * width1 * channel + (x + w) * channel + 0] = 255;
    background[(y - 2) * width1 * channel + (x + w) * channel + 1] = 0;
    background[(y - 2) * width1 * channel + (x + w) * channel + 2] = 0;

    background[(y + height + 2) * width1 * channel + (x + w) * channel + 0] = 255;
    background[(y + height + 2) * width1 * channel + (x + w) * channel + 1] = 0;
    background[(y + height + 2) * width1 * channel + (x + w) * channel + 2] = 0;

    background[(y + height + 1) * width1 * channel + (x + w) * channel + 0] = 255;
    background[(y + height + 1) * width1 * channel + (x + w) * channel + 1] = 0;
    background[(y + height + 1) * width1 * channel + (x + w) * channel + 2] = 0;
}

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

    background[(y + h) * width1 * channel + (x - 2) * channel + 0] = 255;
    background[(y + h) * width1 * channel + (x - 2) * channel + 1] = 0;
    background[(y + h) * width1 * channel + (x - 2) * channel + 2] = 0;

    background[(y + h) * width1 * channel + (x - 1) * channel + 0] = 255;
    background[(y + h) * width1 * channel + (x - 1) * channel + 1] = 0;
    background[(y + h) * width1 * channel + (x - 1) * channel + 2] = 0;

    background[(y + h) * width1 * channel + (x + width + 2) * channel + 0] = 255;
    background[(y + h) * width1 * channel + (x + width + 2) * channel + 1] = 0;
    background[(y + h) * width1 * channel + (x + width + 2) * channel + 2] = 0;

    background[(y + h) * width1 * channel + (x + width + 1) * channel + 0] = 255;
    background[(y + h) * width1 * channel + (x + width + 1) * channel + 1] = 0;
    background[(y + h) * width1 * channel + (x + width + 1) * channel + 2] = 0;
}
```

Step 5: After completing one image, it continues with other images until the end.



IV. Summary

After this project, I have gained new knowledge that not only improves this project but also support many projects in the future as well:

- Cosine Similarity: A metric for measuring similarities. Simple and fast to implement in small and intermediate projects.
- Stepping: The amount of steps shows how much the next area is shifted in the next turn. The smaller this number is, the more accurate can the program to inspect the features of the background image but much slower. On the contrary, larger steps allow faster computing but might result in skipping prominent features. Therefore, the programmer, when use this technique, should find a suitable amount of step to balance between enhancing speed and capturing important features.
- Colored image processing: In general object detection, by converting colored images to grey images, the computing time can be greatly reduced and that is applied in this project. However, in many other areas of computer vision where color plays an important role such as recognising birds, this can make the data losses its crucial features, which is color. For instance, by using the same conversion algorithm above:

$$x = 0.299r + 0.587g + 0.114b \qquad (3)$$

A color value of `(100,0,0)` and `(0,0,262)` would result in the same grey value as such:

$$x_1 = 0.299 * 100 + 0.587 * 0 + 0.114 * 0 = 29.9 \qquad (4)$$

$$x_2 = 0.299 * 0 + 0.587 * 0 + 0.114 * 262 = 29.9 \qquad (5)$$

This causes a red bird to look like a blue in greyscale mode. Therefore, one should consider before use this technique.

References

[1] Had2Know. How to convert rgb to grayscale. 2010.