Advancde computer vision

高等電腦視覺

111618018 自動化碩一 吳祐毅

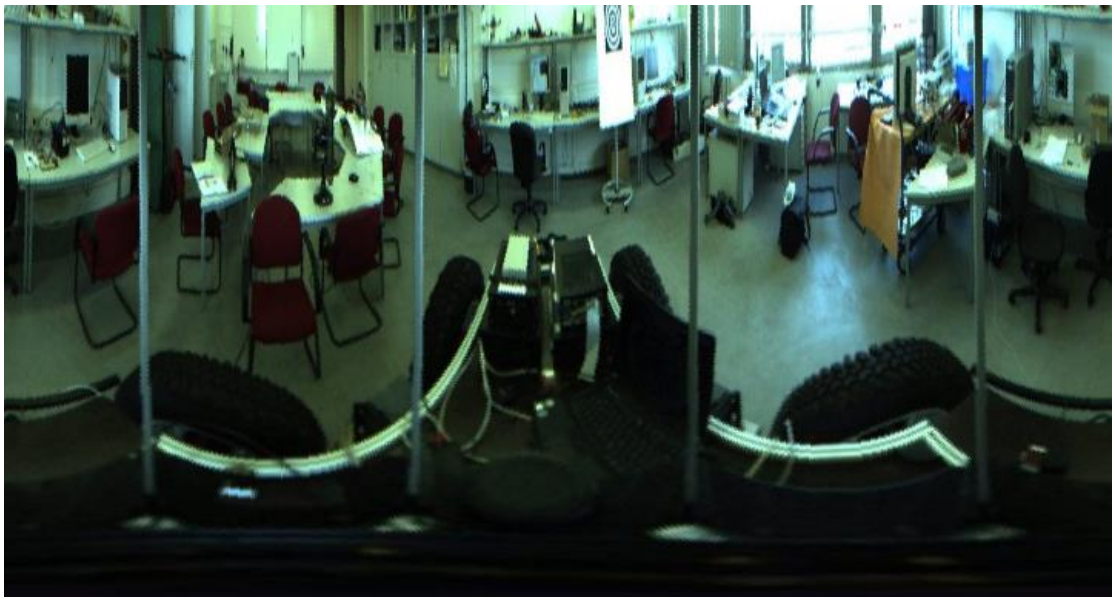
Homework 3

一、 C 語言影像處理(without other library)

1. Rectify the image

將圓柱座標轉換為直角坐標，結果圖中每一個像素點座標，其 x 方向對應到原圖角度 the，y 座標對應到原圖半徑 len，而原圖的圓心其直角坐標為(xc ,yc) = (512,384)。

```
double len, the;
int in_x, in_y;
for(int i = 0; i < out_ysize; i++){
    for(int j = 0; j < out_xsize; j++){
        len = i;
        the = j * (2 * pi) / 720;
        in_x = (int)len * cos(the) + 512;
        in_y = (int)len * sin(the) + 384;
        image_af[(i * out_xsize + j) * 3 + 0] = image[(in_y * in_xsize + in_x) * 3 + 0];
        image_af[(i * out_xsize + j) * 3 + 1] = image[(in_y * in_xsize + in_x) * 3 + 1];
        image_af[(i * out_xsize + j) * 3 + 2] = image[(in_y * in_xsize + in_x) * 3 + 2];
    }
}
```



2. Image stitching

建立一張 900*480 全黑影像，並且將左圖的影像複製到指定的座標位置。

```
//black
for(int i = 0; i < out_ysize; i++){
    for(int j = 0; j < out_xsize; j++){
        image_af[(i * out_xsize + j) * 3 + 2] = 0;
        image_af[(i * out_xsize + j) * 3 + 1] = 0;
        image_af[(i * out_xsize + j) * 3 + 0] = 0;
    }
}

//left
int f_x, f_y;
for(int i = 0; i < in_ysize; i++){
    for(int j = 0; j < in_xsize; j++){
        f_x = j;
        f_y = i + (out_ysize - in_ysize - 80);
        image_af[(f_y * out_xsize + f_x) * 3 + 0] = image_l[((in_ysize - i - 1) * in_xsize + j) * 3 + 2];
        image_af[(f_y * out_xsize + f_x) * 3 + 1] = image_l[((in_ysize - i - 1) * in_xsize + j) * 3 + 1];
        image_af[(f_y * out_xsize + f_x) * 3 + 2] = image_l[((in_ysize - i - 1) * in_xsize + j) * 3 + 0];
    }
}
```

建立轉換前和轉換後的座標(u, v)和(x, y)，以及透過下面公式建立 a~h 的參數。

If we define

$$\Delta x_1 = x_1 - x_2 \quad \Delta x_2 = x_3 - x_2 \quad \Sigma x = x_0 - x_1 + x_2 - x_3$$
$$\Delta y_1 = y_1 - y_2 \quad \Delta y_2 = y_3 - y_2 \quad \Sigma y = y_0 - y_1 + y_2 - y_3$$

then the solution splits into two sub-cases:

(a) $\Sigma x = 0$ and $\Sigma y = 0$. This implies that the xy polygon is a parallelogram, so the mapping is affine, and $a = x_1 - x_0, b = x_2 - x_1, c = x_0, d = y_1 - y_0, e = y_2 - y_1, f = y_0, g = 0, h = 0$.

(b) $\Sigma x \neq 0$ or $\Sigma y \neq 0$ gives a projective mapping:

$$g = \begin{vmatrix} \Sigma x & \Delta x_2 \\ \Sigma y & \Delta y_2 \end{vmatrix} / \begin{vmatrix} \Delta x_1 & \Delta x_2 \\ \Delta y_1 & \Delta y_2 \end{vmatrix}$$

$$h = \begin{vmatrix} \Delta x_1 & \Sigma x \\ \Delta y_1 & \Sigma y \end{vmatrix} / \begin{vmatrix} \Delta x_1 & \Delta x_2 \\ \Delta y_1 & \Delta y_2 \end{vmatrix}$$

$$\begin{aligned} a &= x_1 - x_0 + g x_1 \\ b &= x_3 - x_0 + h x_3 \\ c &= x_0 \\ d &= y_1 - y_0 + g y_1 \\ e &= y_3 - y_0 + h y_3 \\ f &= y_0 \end{aligned} \tag{4}$$

```
//int u0 = 0, v0 = 370, u1 = 0, v1 = 0, u2 = 510, v2 = 0, u3 = 510, v3 = 370;
int u0 = 0, v0 = 0, u1 = 510, v1 = 0, u2 = 510, v2 = 370, u3 = 0, v3 = 370;
//int x0 = 770, y0 = 480, x1 = 250, y1 = 350, x2 = 310, y2 = 10, x3 = 900, y3 = 5;
int x0 = 310, y0 = 10, x1 = 900, y1 = 5, x2 = 770, y2 = 480, x3 = 250, y3 = 350;
int det_x1, det_y1, det_x2, det_y2, sum_x, sum_y;
float a, b, c, d, e, f, g, h, i;
float in_hom[3][3];

det_x1 = x1 - x2;
det_y1 = y1 - y2;
det_x2 = x3 - x2;
det_y2 = y3 - y2;

sum_x = x0 - x1 + x2 - x3;
sum_y = y0 - y1 + y2 - y3;

float g1, g2, h1, h2;
g1 = sum_x * det_y2 - sum_y * det_x2;
g2 = det_x1 * det_y2 - det_y1 * det_x2;
h1 = det_x1 * sum_y - det_y1 * sum_x;
h2 = det_x1 * det_y2 - det_y1 * det_x2;
g = g1 / g2;
h = h1 / h2;
i = 1;
```

接著透過公式計算出圖像扭轉的反轉換矩陣 in_home。

```
a = x1 - x0 + g * x1;
b = x3 - x0 + h * x3;
c = x0;
d = y1 - y0 + g * y1;
e = y3 - y0 + h * y3;
f = y0;

in_hom[0][0] = e * i - f * h;
in_hom[0][1] = f * g - d * i;
in_hom[0][2] = d * h - e * g;
in_hom[1][0] = c * h - b * i;
in_hom[1][1] = a * i - c * g;
in_hom[1][2] = b * g - a * h;
in_hom[2][0] = b * f - c * e;
in_hom[2][1] = c * d - a * f;
in_hom[2][2] = a * e - b * d;
```

掃描所有轉換後圖像的位置，並且計算轉換前的座標，由於圖像扭轉的反轉換矩陣計算出來的座標介於 0~1 之間，而原始圖像其大小為 510*370，所以需要各別乘上 510 和 370 才能對應到原始圖像的座標，在計算完座標之後，需要判斷其座標位置是否有在原始座標裡面。

```
int x_or, y_or;
float x_or1, x_or2, y_or1, y_or2;
float x_ra, y_ra;
for(int i = 0; i < out_ysize; i++){
    for(int j = 0; j < out_xsize; j++){
        x_or1 = in_hom[0][0] * j + in_hom[1][0] * i + in_hom[2][0];
        x_or2 = in_hom[0][2] * j + in_hom[1][2] * i + in_hom[2][2];
        x_or = (int)(x_or1 / x_or2 * in_xsize);
        y_or1 = in_hom[0][1] * j + in_hom[1][1] * i + in_hom[2][1];
        y_or2 = in_hom[0][2] * j + in_hom[1][2] * i + in_hom[2][2];
        y_or = (int)(y_or1 / y_or2 * in_ysize);
        //printf("%f %f\n", x_or1 / x_or2, y_or1 / y_or2);

        if(x_or >= 0 && x_or < 510 && y_or >= 0 && y_or < 370){
            //printf("%d %d %d %d\n",i, j, y_or, x_or);
            image_af[(i * out_xsize + j) * 3 + 0] = image_r[((in_ysize - y_or - 1) * in_xsize + x_or) * 3 + 2];
            image_af[(i * out_xsize + j) * 3 + 1] = image_r[((in_ysize - y_or - 1) * in_xsize + x_or) * 3 + 1];
            image_af[(i * out_xsize + j) * 3 + 2] = image_r[((in_ysize - y_or - 1) * in_xsize + x_or) * 3 + 0];
        }

    }
}
```

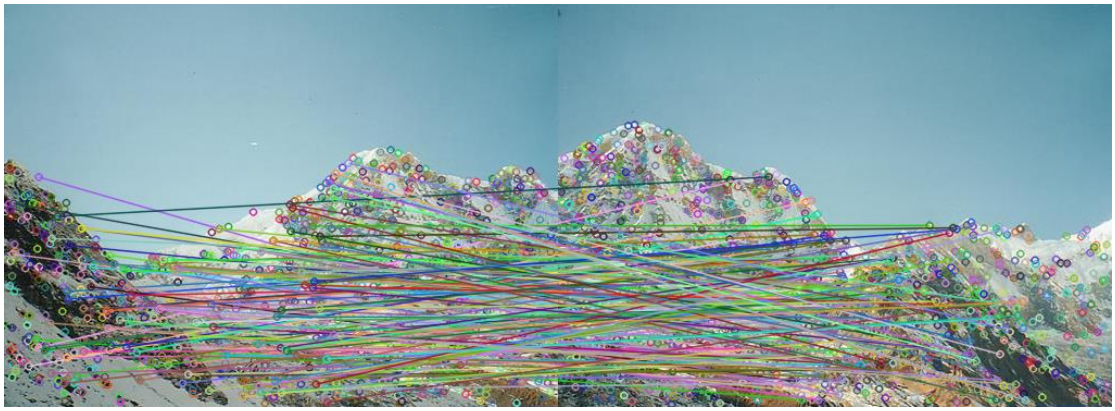
二、 OpenCV 影像處理

先將圖像轉成灰階，接著透過 SIFT 演算法尋找特徵點，最後進行特徵點匹配來計算有多少特徵點。

```
Mat img_1 = imread("left.bmp");
Mat img_r = imread("right.bmp");
imshow("left", img_1);
imshow("right", img_r);
Mat gray_1, gray_r;
cvtColor(img_1, gray_1, COLOR_BGR2GRAY);
cvtColor(img_r, gray_r, COLOR_BGR2GRAY);
//SIFT找特徵點
int Hession = 400;
Ptr<SIFT> sift = SIFT::create();
//檢測關鍵點
vector<KeyPoint> keypoints_1, keypoints_r;
sift -> detect(gray_1, keypoints_1);
sift -> detect(gray_r, keypoints_r);
//計算描述子
Mat descriptor_1, descriptor_r;
sift -> compute(gray_1, keypoints_1, descriptor_1);
sift -> compute(gray_r, keypoints_r, descriptor_r);
//特徵點匹配
vector<vector<DMatch> > matches;
vector<DMatch> good_matches;
Mat img_match;
Ptr<DescriptorMatcher> matcher = DescriptorMatcher::create("BruteForce");
vector<Mat> train_desc(1, descriptor_1);
matcher-> add(train_desc);
matcher -> knnMatch(descriptor_r, matches, 2);
cout << "total match points: " << matches.size() << endl;
```

透過 Lowe's 算法來選取較好的特徵點，並利用這些較好的特徵點來計算出 homography matrix，在將右邊影像透過 warpPerspective 函式來得到變換後的右側圖，以及將左側原圖複製過去就得到兩張拼接後的影像。

```
// Lowe's algorithm,獲取較好的匹配點
for (int i = 0; i < matches.size(); i++)
{
    if (matches[i][0].distance < 0.4 * matches[i][1].distance)   //atio=0. 4：对于准
    {
        good_matches.push_back(matches[i][0]);
    }
}
drawKeypoints(img_1, keypoints_1, descriptor_1, cv::Scalar::all(-1));
drawKeypoints(img_r, keypoints_r, descriptor_r, cv::Scalar::all(-1));
drawMatches(img_1, keypoints_1, img_r, keypoints_r, good_matches, img_match);
imshow("match", img_match);
imwrite("match.bmp", img_match);
//計算H矩陣
vector<cv::Point2f> left_point, right_point;
for (int i = 0; i < good_matches.size(); i++)
{
    right_point.push_back(keypoints_r[good_matches[i].queryIdx].pt);
    left_point.push_back(keypoints_1[good_matches[i].trainIdx].pt);
}
Mat H = findHomography(right_point, left_point);
Mat stitchedImage;
warpPerspective(img_r, stitchedImage, H, Size(img_1.cols + img_r.cols, img_r.rows));
imshow("右圖視角", stitchedImage);
Mat half(stitchedImage, Rect(0, 0, img_1.cols, img_1.rows));
img_1.copyTo(half);
imshow("result", stitchedImage);
imwrite("result.bmp", stitchedImage);
```



特徵點匹配

右側圖變換後

兩張圖拼接後結果

三、 Bonus

先計算出左圖、右圖、風格轉換圖各別的 rgb 0~255 的強度分布，也就是得到機率密度函數(PDF)，接著再去計算各別累積分布函數(CDF)。

```
//transfer style
int in_his_l[3][256], in_his_r[3][256], sty_his[3][256];
float pr_l[3][256], pr_r[3][256], pz_sty[3][256];
float pr_l_total[3][256], pr_r_total[3][256], pz_sty_total[3][256];
for(int i = 0; i < 3; i++){
    for(int j = 0; j < 256; j++){
        in_his_l[i][j] = 0;
        in_his_r[i][j] = 0;
        sty_his[i][j] = 0;
        pr_l[i][j] = 0;
        pr_r[i][j] = 0;
        pz_sty[i][j] = 0;
        pz_sty_total[i][j] = 0;
    }
}
for(int i = 0; i < in_ysize; i++){
    for(int j = 0; j < in_xsize; j++){
        in_his_l[0][image_l[(i * in_xsize + j) * 3 + 0]] ++;
        in_his_l[1][image_l[(i * in_xsize + j) * 3 + 1]] ++;
        in_his_l[2][image_l[(i * in_xsize + j) * 3 + 2]] ++;
        in_his_r[0][image_r[(i * in_xsize + j) * 3 + 0]] ++;
        in_his_r[1][image_r[(i * in_xsize + j) * 3 + 1]] ++;
        in_his_r[2][image_r[(i * in_xsize + j) * 3 + 2]] ++;
    }
}
```

```
for(int i = 0; i < sty_ysize; i++){
    for(int j = 0; j < sty_xsize; j++){
        sty_his[0][image_sty[(i * sty_xsize + j) * 4 + 0]] ++;
        sty_his[1][image_sty[(i * sty_xsize + j) * 4 + 1]] ++;
        sty_his[2][image_sty[(i * sty_xsize + j) * 4 + 2]] ++;
    }
}

for(int i = 0; i < 3; i++){
    for(int j = 0; j < 256; j++){
        pr_l[i][j] = (float)in_his_l[i][j] / (in_xsize * in_ysize);
        pr_r[i][j] = (float)in_his_r[i][j] / (in_xsize * in_ysize);
        pz_sty[i][j] = (float)sty_his[i][j] / (sty_xsize * sty_ysize);
    }
}

for(int i = 0; i < 3; i++){
    for(int j = 0; j < 256; j++){
        for(int k = 0; k <= j; k++){
            pr_l_total[i][j] = (float)pr_l_total[i][j] + (float)pr_l[i][k];
            pr_r_total[i][j] = (float)pr_r_total[i][j] + (float)pr_r[i][k];
            pz_sty_total[i][j] = (float)pz_sty_total[i][j] + (float)pz_sty[i][k];
        }
    }
}
```

接著再去比對左圖和右圖中每一個(CDF)的值與風格轉換圖中(CDF)的值去進行比較，並找出最小的值來當作其轉換後所對應的像素值。

```
float diff_l[256], diff_r[256], min_diff_l, min_diff_r;
int index_l, index_r;
for(int i = 0; i < 3; i++){
    for(int j = 0; j < 256; j++){
        for(int k = 0; k < 256; k++){
            diff_l[k] = pr_l_total[i][j] -  pz_sty_total[2 - i][k];
            if(diff_l[k] < 0){
                diff_l[k] = diff_l[k] * -1;
            }
            diff_r[k] = pr_r_total[i][j] -  pz_sty_total[2 - i][k];
            if(diff_r[k] < 0){
                diff_r[k] = diff_r[k] * -1;
            }
        }
        min_diff_l = diff_l[0];
        min_diff_r = diff_r[0];
        index_l = 0;
        index_r = 0;
        for(int k = 0; k < 256; k++){
            if(min_diff_l > diff_l[k]){
                min_diff_l = diff_l[k];
                index_l = k;
            }
            if(min_diff_r > diff_r[k]){
                min_diff_r = diff_r[k];
                index_r = k;
            }
        }
```

```
        }
        for(int y = 0; y < in_ysize; y++){
            for(int x = 0; x < in_xsize; x++){
                if(image_l[(y * in_xsize + x) * 3 + i] == j){
                    image_l[(y * in_xsize + x) * 3 + i] = index_l;
                }
                if(image_r[(y * in_xsize + x) * 3 + i] == j){
                    image_r[(y * in_xsize + x) * 3 + i] = index_r;
                }
            }
        }
    }
}
```