Homework #2

ECE661

Name:Chengzhang Zhong

Email:zhongc@purdue.edu

Homographies calculation

To compute the homogenous to map a 3-vector in world plane in to image, the equation can be written as following:

$$\boldsymbol{X_i} = \boldsymbol{H} * \boldsymbol{X_w}$$

which can also be represented in matrix format written as:

$$\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ 1 \end{bmatrix}$$

Also, we can choose $x_i^{'}$ and $y_i^{'}$ to represent points in 2D space of real. Then, it is clear the following equations hold:

$$x_i^{'} = \frac{x_i}{z_i} = \frac{h_{11}x_w + h_{12}y_w + h_{13}}{h_{31}x_w + h_{32}y_w + h_{33}}$$
$$y_i^{'} = \frac{y_i}{z_i} = \frac{h_{21}x_w + h_{22}y_w + h_{23}}{h_{31}x_w + h_{32}y_w + h_{33}}$$

Replace h_{33} with 1 and after doing simplification:

$$h_{11}x_w + h_{12}y_w + h_{13} - h_{31}x_i^{'}x_w - h_{32}x_i^{'}y_w = x_i^{'}$$

$$h_{21}x_w + h_{22}y_w + h_{23} - h_{31}y_i^{'}x_w - h_{32}y_i^{'}y_w = y_i^{'}$$

Therefore, this question has 8 unknowns $[h_{11}\ h_{12}\ h_{13}\ h_{21}\ h_{22}\ h_{23}\ h_{31}\ h_{32}]$. To solve this problem, it is necessary to provide 8 equations about these unknowns, which corresponds to 4 points. If we represent them in a matrix form $Ax = b$:

$$\begin{bmatrix} x_w^1 & y_w^1 & 1 & 0 & 0 & 0 & -x_i^{'}x_w^1 & -x_i^{'}y_w^1 \\ 0 & 0 & 0 & x_w^1 & y_w^1 & 1 & -y_i^{'}x_w^1 & -y_i^{'}y_w^1 \\ x_w^2 & y_w^2 & 1 & 0 & 0 & 0 & -x_i^{'}x_w^2 & -x_i^{'}y_w^2 \\ 0 & 0 & 0 & x_w^2 & y_w^2 & 1 & -y_i^{'}x_w^2 & -y_i^{'}y_w^2 \\ x_w^3 & y_w^3 & 1 & 0 & 0 & 0 & -x_i^{'}x_w^3 & -x_i^{'}y_w^3 \\ 0 & 0 & 0 & x_w^3 & y_w^3 & 1 & -y_i^{'}x_w^3 & -y_i^{'}y_w^3 \\ x_w^4 & y_w^4 & 1 & 0 & 0 & 0 & -x_i^{'}x_w^4 & -x_i^{'}y_w^4 \\ 0 & 0 & 0 & x_w^4 & y_w^4 & 1 & -y_i^{'}x_w^4 & -y_i^{'}y_w^4 \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = \begin{bmatrix} x_i^{'1} \\ y_i^{'1} \\ x_i^{'2} \\ y_i^{'2} \\ x_i^{'3} \\ y_i^{'3} \\ x_i^{'4} \\ y_i^{'4} \end{bmatrix}$$

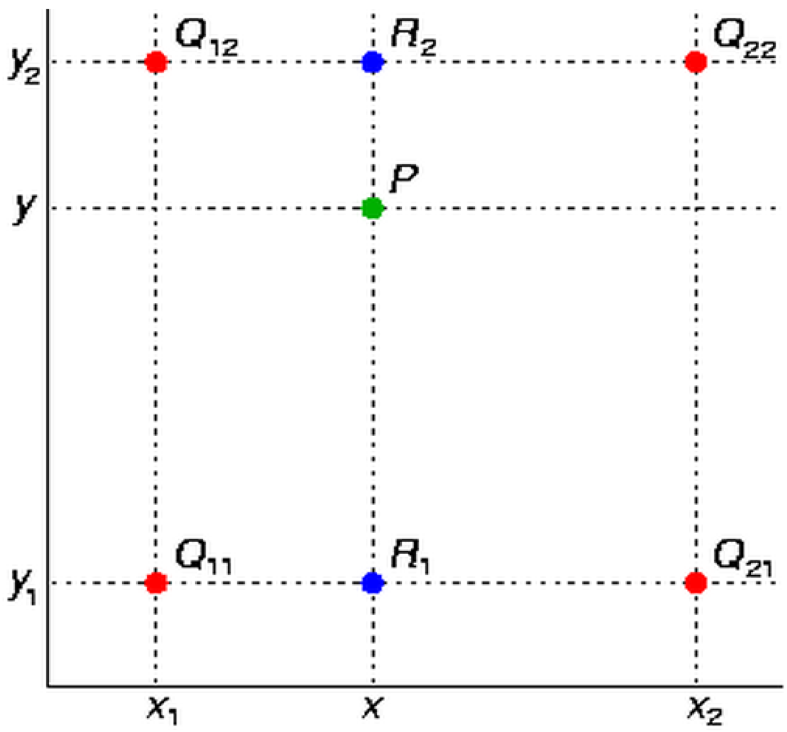
From here, the equation can be solved as $x = A^{\dagger}b$,where $A^{\dagger}$ is the pseudo-inverse.

bilinear interpolation

The method I took to map world plane to image plane works in an inverse way.

- First, find the inverse of homogeneous $\boldsymbol{H}$ as $\boldsymbol{H^{-1}}$**.** For the convenience in calculation, I selected an rectangular shape which encompasses the P,Q,R,S region in image plane.
- Second, the mapping process works by selecting every points inside the rectangular region, multiply it with $\boldsymbol{H^{-1}}$ to get the its corresponding location in world plane.
- Finally, use the pixel value of the point in world plane to replace the corresponding pixel value in image plane.

However, in the second step of multiplication with $\boldsymbol{H^{-1}}$, it usually generate pixel locations with float numbers. To solve this issue, I took bilinear interpolation method. Instead of using a pixel location P which doesn't exist, the method tends to calculate the nearest four points.



$$f(R_1) = \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21})$$

$$f(R_2) = \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22})$$

$$f(P) = \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2)$$

where $f(P)$ is the final pixel value filled into image plane.

Task 1.

See points used in these figures from the code

Figure 1(a).

Figure 1(b).

Figure 1(c).

Task 2.

See points used in these figures from the code

Figure 2(a). original image of "1.jpg"

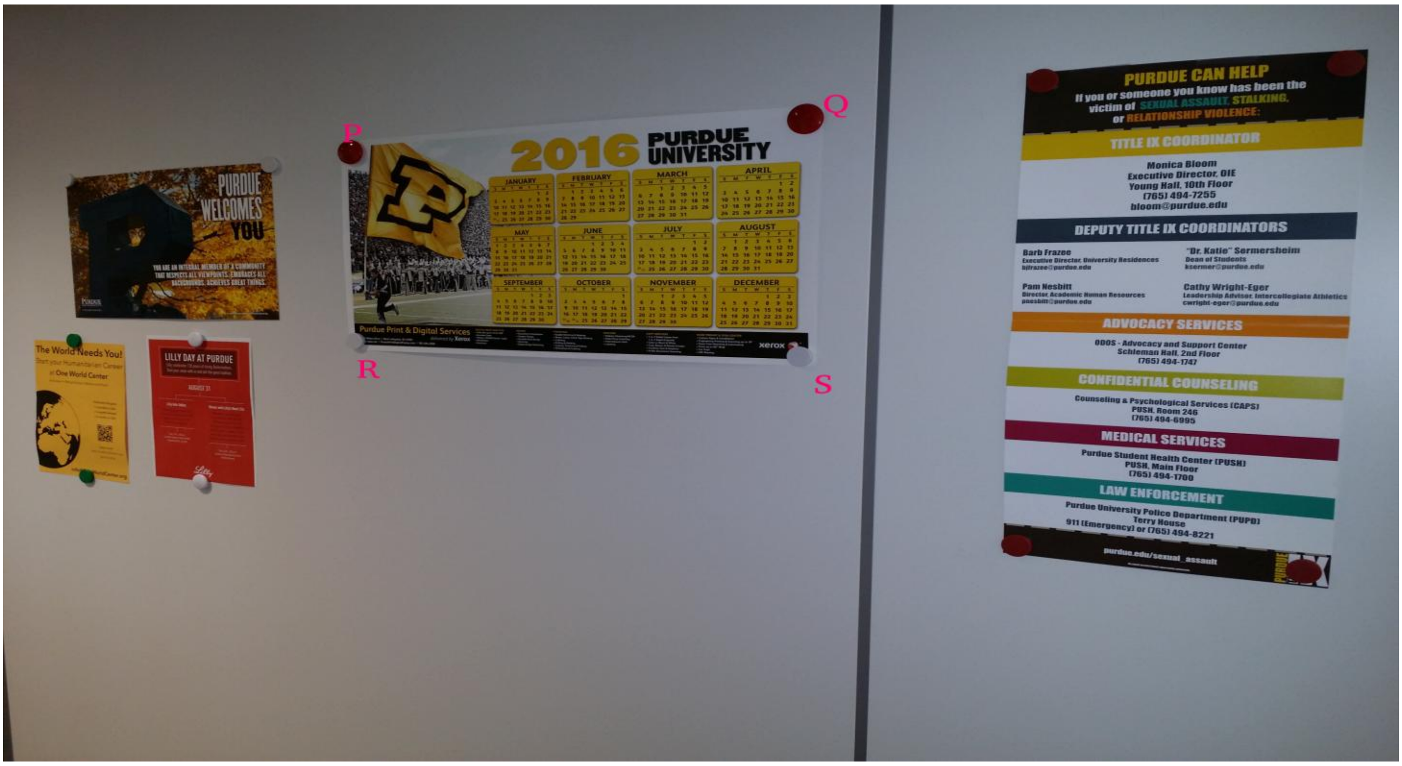


Figure 2(b). original image of "3.jpg"

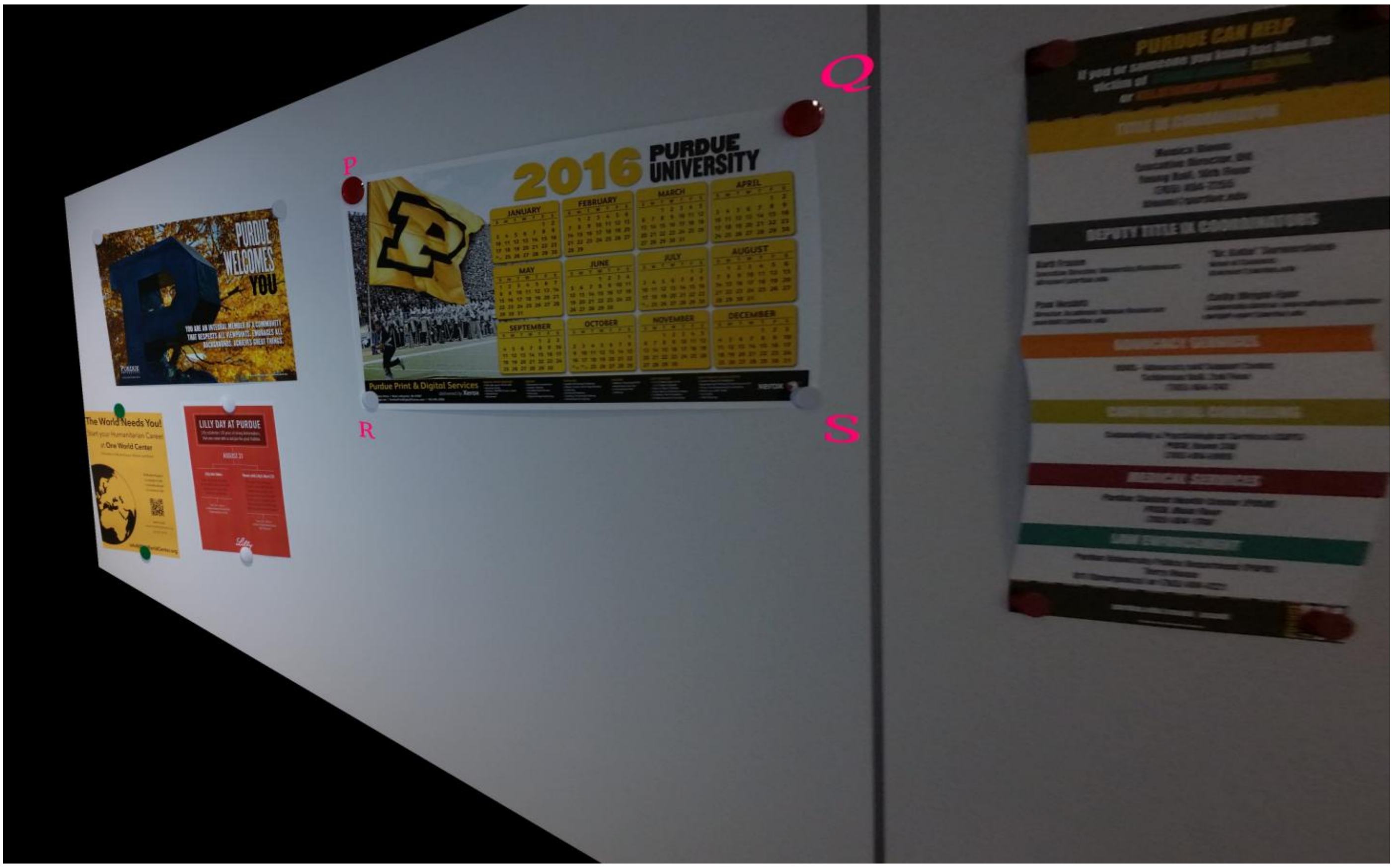


Figure 2(c). "1.jpg" mapped into "3.jpg"

Task 3

See points used in these figures from the code

Figure 3(a). the first image from my own

Figure 3(b). the second image from my own

Figure 3(c). the third image from my own

Figure 3(d). my own person image

Figure 3(e). Map my personal image into my first image

Figure 3(f). Map my personal image into my second image

Figure 3(g). Map my personal image into my third image

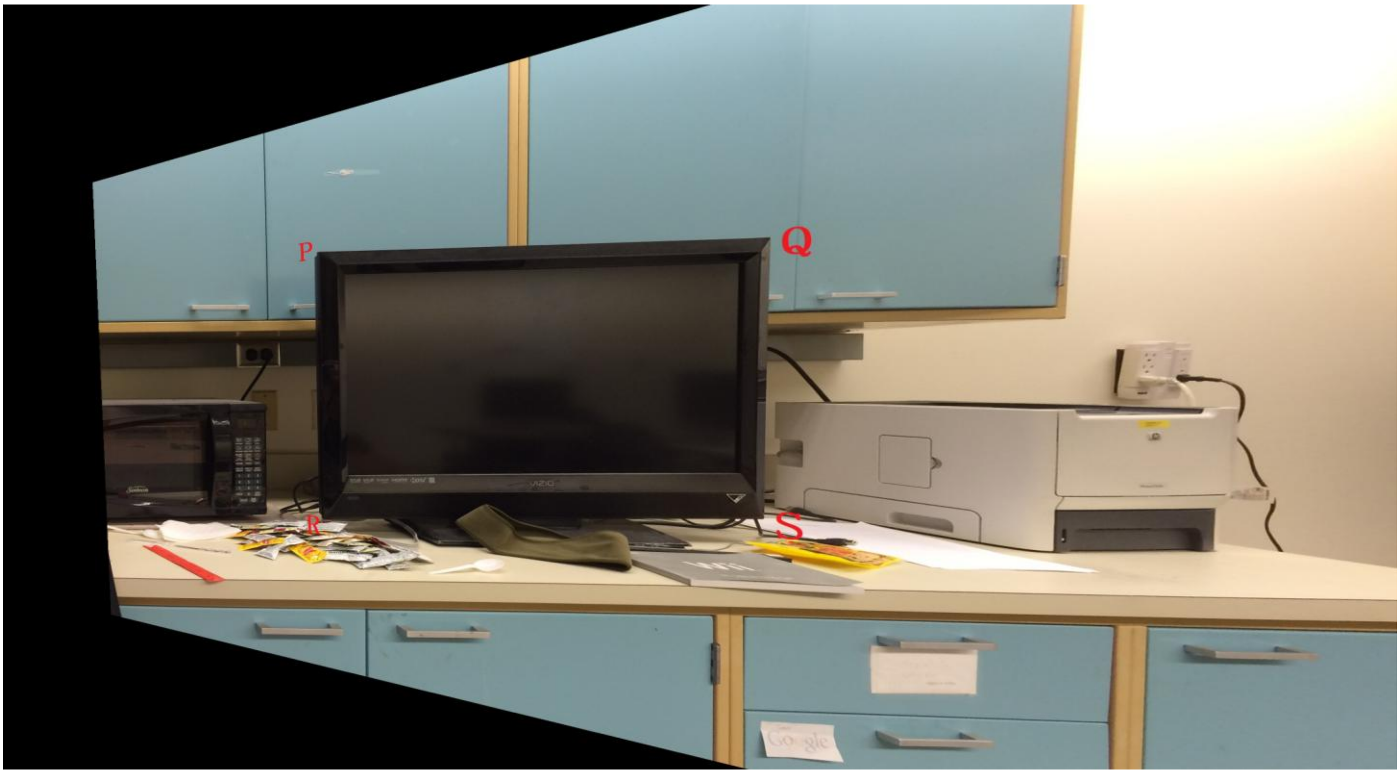


Figure 3(h). Map my first image into my third image

Source code(in c++)

(a). Main functions:

```
int main() {
    Mat face = imread("E:\\2016Fall\\661\\HW2\\Seinfeld.jpg",1);
    Mat one_a = imread("E:\\2016Fall\\661\\HW2\\1.jpg",1);
    Mat one_a_rep = one_a.clone();
    Mat one_b = imread("E:\\2016Fall\\661\\HW2\\2.jpg",1);
    Mat one_b_rep = one_b.clone();
    Mat one_c = imread("E:\\2016Fall\\661\\HW2\\3.jpg",1);
    Mat one_c_rep = one_c.clone();


/******************************
Task 1. Mapping face image into figure 1.a, figure 1.b, figure 1.c

******************************/
    //For Seinfeld.jpg image
    Point2f P_f_a = Point2f(0,0); // P -- 0
    Point2f Q_f_a = Point2f(2560,0);// Q -- 1
    Point2f R_f_a = Point2f(0,1536); //  R -- 2
    Point2f S_f_a = Point2f(2560,1536);// S -- 3

    //For 1.jpg
    Point2f P_i_a = Point2f(2140,448); // P -- 0
    Point2f Q_i_a = Point2f(3304,552);// Q -- 1
    Point2f R_i_a = Point2f(2152,1500); //  R -- 2
    Point2f S_i_a = Point2f(3304,1344);// S -- 3
    //For 2.jpg
    Point2f P_i_b = Point2f(1612,816); // P -- 0
    Point2f Q_i_b = Point2f(2976,784);// Q -- 1
    Point2f R_i_b = Point2f(1648,1576); //  R -- 2
    Point2f S_i_b = Point2f(2972,1500);// S -- 3
    //For 3.jpg
    Point2f P_i_c = Point2f(1008,580); // P -- 0
    Point2f Q_i_c = Point2f(2388,428);// Q -- 1
    Point2f R_i_c = Point2f(1036,1396); //  R -- 2
    Point2f S_i_c = Point2f(2380,1472);// S -- 3

    Mat H_inverse_a;
    float height_length_a;
    float width_length_a;

    Mat H_inverse_b;
    float height_length_b;
    float width_length_b;
```

```
    Mat H_inverse_c;
    float height_length_c;
    float width_length_c;

    Calc_homography( P_f_a, Q_f_a, R_f_a,  S_f_a,  P_i_a, Q_i_a, R_i_a, S_i_a, H_inverse_a);
    Calc_homography( P_f_a, Q_f_a, R_f_a,  S_f_a,  P_i_b, Q_i_b, R_i_b, S_i_b, H_inverse_b);
    Calc_homography( P_f_a, Q_f_a, R_f_a,  S_f_a,  P_i_c, Q_i_c, R_i_c, S_i_c, H_inverse_c);


    max_area(&height_length_a, &width_length_a, P_i_a, Q_i_a, R_i_a, S_i_a);
    max_area(&height_length_b, &width_length_b, P_i_b, Q_i_b, R_i_b, S_i_b);
    max_area(&height_length_c, &width_length_c, P_i_c, Q_i_c, R_i_c, S_i_c);

    face2target(H_inverse_a, height_length_a, width_length_a,  P_i_a, P_f_a, S_f_a, face, one_a_rep);
    face2target(H_inverse_b, height_length_b, width_length_b,  P_i_b, P_f_a, S_f_a, face, one_b_rep);
    face2target(H_inverse_c, height_length_c, width_length_c,  Point2f(1008,428), P_f_a, S_f_a, face,
one_c_rep);
    imwrite( "E:\\2016Fall\\661\\HW2\\tryyyy.jpg",one_a_rep);
    imwrite( "E:\\2016Fall\\661\\HW2\\step_b.jpg",one_b_rep);
    imwrite( "E:\\2016Fall\\661\\HW2\\step_c.jpg",one_c_rep);

/*****************************
Task 2. Homographies between figure 1.a, figure 1.b, figure 1.c

*****************************/
    Mat H_ab;
    Mat H_bc;
    Mat H_ac;
    Mat H_inv_ac;
    float height_length_ac = static_cast<float> (one_c.rows);
    float width_length_ac = static_cast<float> (one_c.cols);
    Mat task2_img(3096,4128, CV_8UC3, Scalar(0,0,0));

    Calc_homo_task2( P_i_a, Q_i_a, R_i_a, S_i_a, P_i_b, Q_i_b, R_i_b, S_i_b, H_ab);
    Calc_homo_task2( P_i_b, Q_i_b, R_i_b, S_i_b,  P_i_c, Q_i_c, R_i_c, S_i_c, H_bc);
    H_ac = H_ab * H_bc;
    H_inv_ac = H_ac.inv();

    face2target(H_inv_ac, height_length_ac, width_length_ac,  Point2f(0,0), Point2f(0,0),
Point2f(4128-1,3096 - 1), one_a, task2_img);
    imwrite( "E:\\2016Fall\\661\\HW2\\task2.jpg",task2_img);


/*****************************************************
Task 3. Use you own image to repeat task 1 and 2
*****************************************************/
    Mat person = imread("E:\\2016Fall\\661\\HW2\\person.jpg",1);
    Mat my1 = imread("E:\\2016Fall\\661\\HW2\\my1.jpg",1);
    Mat my1_rep = my1.clone();
```

```
Mat my2 = imread("E:\\2016Fall\\661\\HW2\\my2.jpg",1);
Mat my2_rep = my2.clone();
Mat my3 = imread("E:\\2016Fall\\661\\HW2\\my3.jpg",1);
Mat my3_rep = my3.clone();

 ty =  type2str( my3.type() );
printf("Matrix: %s %dx%d \n", ty.c_str(), my3.cols, my3.rows );


//For person image
Point2f P_p_a = Point2f(0,0); // P -- 0
Point2f Q_p_a = Point2f(2560,0);// Q -- 1
Point2f R_p_a = Point2f(0,1440); //  R -- 2
Point2f S_p_a = Point2f(2560,1440);// S -- 3

//For my first image
Point2f P_i_1 = Point2f(1112,600); // P -- 0
Point2f Q_i_1 = Point2f(2336,772);// Q -- 1
Point2f R_i_1 = Point2f(1092,1768); //  R -- 2
Point2f S_i_1 = Point2f(2316,1636);// S -- 3
//For my second image
Point2f P_i_2 = Point2f(700,544); // P -- 0
Point2f Q_i_2 = Point2f(2664,564);// Q -- 1
Point2f R_i_2 = Point2f(732,1856); //  R -- 2
Point2f S_i_2 = Point2f(2664,1824);// S -- 3
//For my third image
Point2f P_i_3 = Point2f(948,820); // P -- 0
Point2f Q_i_3 = Point2f(2116,656);// Q -- 1
Point2f R_i_3 = Point2f(952,1608); //  R -- 2
Point2f S_i_3 = Point2f(2112,1712);// S -- 3

Mat H_inverse_1;
float height_length_1;
float width_length_1;

Mat H_inverse_2;
float height_length_2;
float width_length_2;

Mat H_inverse_3;
float height_length_3;
float width_length_3;

Calc_homography( P_p_a, Q_p_a, R_p_a,  S_p_a,  P_i_1, Q_i_1, R_i_1, S_i_1, H_inverse_1);
Calc_homography( P_p_a, Q_p_a, R_p_a,  S_p_a,  P_i_2, Q_i_2, R_i_2, S_i_2, H_inverse_2);
Calc_homography( P_p_a, Q_p_a, R_p_a,  S_p_a,  P_i_3, Q_i_3, R_i_3, S_i_3, H_inverse_3);


max_area(&height_length_1, &width_length_1, P_i_1, Q_i_1, R_i_1, S_i_1);
```

```
    max_area(&height_length_2, &width_length_2, P_i_2, Q_i_2, R_i_2, S_i_2);
    max_area(&height_length_3, &width_length_3, P_i_3, Q_i_3, R_i_3, S_i_3);

    face2target(H_inverse_1, height_length_1, width_length_1,  P_i_1, P_p_a, S_p_a, person, my1_rep);
    face2target(H_inverse_2, height_length_2, width_length_2,  P_i_2, P_p_a, S_p_a, person, my2_rep);
    face2target(H_inverse_3, height_length_3, width_length_3,  Point2f(948,656), P_p_a, S_p_a, person,
my3_rep);
    imwrite( "E:\\2016Fall\\661\\HW2\\task3_my1.jpg",my1_rep);
    imwrite( "E:\\2016Fall\\661\\HW2\\task3_my2.jpg",my2_rep);
    imwrite( "E:\\2016Fall\\661\\HW2\\task3_my3.jpg",my3_rep);


    Mat H_12;
    Mat H_23;
    Mat H_13;
    Mat H_inv_13;
    float height_length_13 = static_cast<float> (my3.rows);
    float width_length_13 = static_cast<float> (my3.cols);
    Mat task3_my1tomy3(my3.rows,my3.cols, CV_8UC3, Scalar(0,0,0));

    Calc_homo_task2( P_i_1, Q_i_1, R_i_1, S_i_1, P_i_2, Q_i_2, R_i_2, S_i_2, H_12);
    Calc_homo_task2( P_i_2, Q_i_2, R_i_2, S_i_2,  P_i_3, Q_i_3, R_i_3, S_i_3, H_23);
    H_13 = H_12 * H_23;
    H_inv_13 = H_13.inv();

    face2target(H_inv_13, height_length_13, width_length_13,  Point2f(0,0), Point2f(0,0), Point2f(3264
- 1,2448 - 1), my1, task3_my1tomy3);
    imwrite( "E:\\2016Fall\\661\\HW2\\task3_my1tomy3.jpg",task3_my1tomy3);
    return 1;
}
```

(b). sub-functions

```
/***********************************
Calculate max and min between two floats
***********************************/
float max(float num1, float num2) {
    return (num1 >= num2)?num1:num2;
}
float min(float num1, float num2) {
    return (num1 <= num2)?num1:num2;
}


/***********************************
Mapping face image to the target image
***********************************/
void face2target(Mat H, float height, float width, Point2f origin_pt, Point2f LT, Point2f BR, Mat face,
```

```
Mat &target){
    for (float i = origin_pt.y ; i < origin_pt.y +height; i++) {
        for (float j = origin_pt.x ; j < origin_pt.x  + width; j++) {

            Mat start_pt = Mat (3,1,CV_32F); //the point in the frame image
            start_pt.at<float>(0,0) = j;
            start_pt.at<float>(1,0) = i;
            start_pt.at<float>(2,0) = 1;
            Mat end_pt = H * start_pt;

            float x_coord = end_pt.at<float>(0,0)/end_pt.at<float>(2,0);
            float y_coord = end_pt.at<float>(1,0)/end_pt.at<float>(2,0);

            if (x_coord < LT.x || x_coord >BR.x || y_coord < LT.y || y_coord > BR.y) {
                continue; //the case that point we want is not inside the face image
            }
            else {
                bilinear(j,i,x_coord,y_coord,face,target);
            }
        }
    }
}

/***************************************
Find the height and width of a region which covers PQRS in the target image
--height - the result height
--width - the result width
-- P_i, Q_i, R_i, S_i - P,Q,R,S 2D points from target image
***************************************/
void max_area(float *height, float *width, Point2f P_i,Point2f Q_i,Point2f R_i,Point2f S_i) {
    *height = max(R_i.y - P_i.y,S_i.y - Q_i.y);
    *width = max(Q_i.x - P_i.x, S_i.x - R_i.x);
}
/***************************************
Calcluate and return homegraphy matrix for task 2
-- P_f, Q_f, R_f, S_f - P,Q,R,S 2D points from face image
-- P_i, Q_i, R_i, S_i - P,Q,R,S 2D points from target image
***************************************/
void Calc_homo_task2(Point2f P_f,Point2f Q_f,Point2f R_f, Point2f S_f, Point2f P_i,Point2f Q_i,Point2f
R_i,Point2f S_i, Mat &H) {
    vector<Point2f> face_corners;
    vector<Point2f> face_mid;
    vector<Point2f> one_a_corners;
    vector<Point2f> one_a_mid;
    //Create matrix representation
    Mat large = Mat(8,8,CV_32F); //large matrix A
    Mat small = Mat(8,1,CV_32F); //small matrix b
```

```
    //Input 4 points for each figure
    face_corners.push_back(P_f); // P -- 0
    face_corners.push_back(Q_f);// Q -- 1
    face_corners.push_back(R_f); // R -- 2
    face_corners.push_back(S_f);//S -- 3

    one_a_corners.push_back(P_i); // P -- 0
    one_a_corners.push_back(Q_i);// Q -- 1
    one_a_corners.push_back(R_i); // R -- 2
    one_a_corners.push_back(S_i);//S -- 3

    //Intialize matrix
    int pt_num = large.rows/2; //number of points to calculate
    for (int i = 0; i < pt_num * 2; i++) {
        if (i % 2 == 0) { // write the row in x-point's format
                large.at<float>(i,0) = face_corners[i/2].x;
                large.at<float>(i,1) = face_corners[i/2].y;
                large.at<float>(i,2) = 1;
                large.at<float>(i,3) = 0;
                large.at<float>(i,4) = 0;
                large.at<float>(i,5) = 0;
                large.at<float>(i,6) = -face_corners[i/2].x * one_a_corners[i/2].x;
                large.at<float>(i,7) = -face_corners[i/2].y * one_a_corners[i/2].x;
                small.at<float>(i,0) = one_a_corners[i/2].x;
            }

        if (i % 2 == 1) { //write the row in y-point's format
                large.at<float>(i,0) = 0;
                large.at<float>(i,1) = 0;
                large.at<float>(i,2) = 0;
                large.at<float>(i,3) = face_corners[(i-1)/2].x;
                large.at<float>(i,4) = face_corners[(i-1)/2].y;
                large.at<float>(i,5) = 1;
                large.at<float>(i,6) = -face_corners[(i-1)/2].x * one_a_corners[(i-1)/2].y;
                large.at<float>(i,7) = -face_corners[(i-1)/2].y * one_a_corners[(i-1)/2].y;
                small.at<float>(i,0) = one_a_corners[(i-1)/2].y;
            }
        }

    //calculate H matrix
    Mat H_mat = large.inv(DECOMP_SVD) * small;
    Mat h33 = Mat (1,1,CV_32F);
    h33.at<float>(0,0) = 1;
    H_mat.push_back(h33); //add h33 as 1
    H =  H_mat.reshape(0,3);
}
```

```
/****************************************
Calcluate and return the inversed homegraphy matrix
-- P_f, Q_f, R_f, S_f - P,Q,R,S 2D points from face image
-- P_i, Q_i, R_i, S_i - P,Q,R,S 2D points from target image
****************************************/
void Calc_homography(Point2f P_f,Point2f Q_f,Point2f R_f, Point2f S_f, Point2f P_i,Point2f Q_i,Point2f
R_i,Point2f S_i, Mat &H_inverse) {
    vector<Point2f> face_corners;
    vector<Point2f> face_mid;
    vector<Point2f> one_a_corners;
    vector<Point2f> one_a_mid;
    //Create matrix representation
    Mat large = Mat(8,8,CV_32F); //large matrix A
    Mat small = Mat(8,1,CV_32F); //small matrix b

    //labeled as Figure 2 in homework
    //Input 8 points for each figure
    face_corners.push_back(P_f); // P -- 0
    face_corners.push_back(Q_f);// Q -- 1
    face_corners.push_back(R_f); // R -- 2
    face_corners.push_back(S_f);//S -- 3

    one_a_corners.push_back(P_i); // P -- 0
    one_a_corners.push_back(Q_i);// Q -- 1
    one_a_corners.push_back(R_i); // R -- 2
    one_a_corners.push_back(S_i);//S -- 3

    //Intialize matrix
    int pt_num = large.rows/2; //number of points to calculate
    for (int i = 0; i < pt_num * 2; i++) {
        if (i % 2 == 0) { // write the row in x-point's format
                large.at<float>(i,0) = face_corners[i/2].x;
                large.at<float>(i,1) = face_corners[i/2].y;
                large.at<float>(i,2) = 1;
                large.at<float>(i,3) = 0;
                large.at<float>(i,4) = 0;
                large.at<float>(i,5) = 0;
                large.at<float>(i,6) = -face_corners[i/2].x * one_a_corners[i/2].x;
                large.at<float>(i,7) = -face_corners[i/2].y * one_a_corners[i/2].x;
                small.at<float>(i,0) = one_a_corners[i/2].x;
            }

        if (i % 2 == 1) { //write the row in y-point's format
                large.at<float>(i,0) = 0;
                large.at<float>(i,1) = 0;
                large.at<float>(i,2) = 0;
                large.at<float>(i,3) = face_corners[(i-1)/2].x;
                large.at<float>(i,4) = face_corners[(i-1)/2].y;
```

```
                large.at<float>(i,5) = 1;
                large.at<float>(i,6) = -face_corners[(i-1)/2].x * one_a_corners[(i-1)/2].y;
               large.at<float>(i,7) = -face_corners[(i-1)/2].y * one_a_corners[(i-1)/2].y;
                small.at<float>(i,0) = one_a_corners[(i-1)/2].y;
            }
        }

    //calculate H matrix
    Mat H_mat = large.inv(DECOMP_SVD) * small;
    Mat h33 = Mat (1,1,CV_32F);
    h33.at<float>(0,0) = 1;
    H_mat.push_back(h33); //add h33 as 1
    Mat H =  H_mat.reshape(0,3);
    H_inverse = H.inv();
}


/****************************************
bilinear interpolation fucntion:
--col - column in the dest image
--row - row in the dest image
--x0 - corresponding col location in the face image
--y0 - corresponding row location in the face image
--source - face image
--dest - the image which the face mapped to
****************************************/
void bilinear(float col, float row,float x0,float y0,Mat source, Mat &dest) {
    int x1 = int(x0);
    int x2 = x1 + 1;
    int y1 = int(y0);
    int y2 = y1+1;
    float wx1;
    float wx2;
    float wy1;
    float wy2;
    int height = source.rows;
    int width = source.cols;
    if (x1 < 0 || x1 >= width || x2 < 0 || x2 >= width || y1 < 0 || y1 >= height || y2 < 0 || y2 >= height)
{ //if go across image boundary
        for (int k = 0; k < 3; k++) {// k = 0, blue channel. k = 1, green channel. k = 2, red channel
            float result = 0;
            dest.at<Vec3b>(row,col)[k] = static_cast<uchar>(result);

        }
    }
    else {
            wx1 = x0 - x1;
            wx2 = 1 - wx1;
            wy1 = y0 - y1;
```

```
            wy2 = 1 - wy1;
            float r1 = wx1 * wy1;
            float r2 = wx2 * wy1;
            float r3 = wx2 * wy2;
            float r4 = wx1 * wy2;

            for (int k = 0; k < 3; k++) {// k = 0, blue channel. k = 1, green channel. k = 2, red channel

            uchar a = source.at<Vec3b>(y2,x2)[k];
            uchar b = source.at<Vec3b>(y2,x1)[k];
            uchar c = source.at<Vec3b>(y1,x1)[k];
            uchar d = source.at<Vec3b>(y1,x2)[k];



            float result =
static_cast<float>(a)*r1+static_cast<float>(b)*r2+static_cast<float>(c)*r3+static_cast<float>(d)*r4;
            if(result >255) {
                result = 255;
            } else if (result < 0) {
                result = 0;
            }
            dest.at<Vec3b>(row,col)[k] = static_cast<uchar>(result);
            }
    }

}
```