Part 1:
1.1) We were supposed to implement inverse warping technique and get the warped image . We were successfully able to do so by calculating the coordinates in the warped image and simply copying the value of pixels from the respective image. We also calculated the inverse of the transformation matrix which is required for inverse Warping.

Resultant Image:

1.2) We were supposed to write a function to calculate the ProjectiveTransformation Matrix given 4 sets of coordinates from 2 images. We were successfully able to do so by solving a system of linear equations mentioned on Page 200 of Burger Vol.2 . After that we were supposed to use the calculated transformation matrix and get the 3rd image.

Resultant Image:

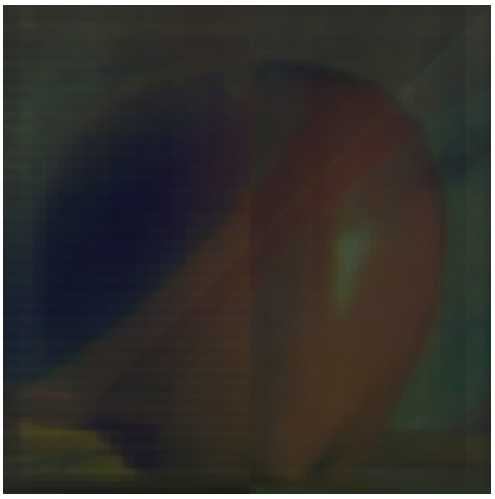
1.3) In this part , we were supposed to find out the transformation matrix of the poster image and then calculate the warped image using the transformation matrix. To calculate the transformation matrix, we needed 8 sets of coordinates including the coordinates of the 4 corners of the billboard.
After that, we would simply add up the 2 images such that the values superimpose each other and then normalize the new poster image.
We were able to find the coordinates of the billboard but for some reason when we tried to calculate the transformation matrix, we were not getting promising results.
USAGE : the images required should be present in the "images/part1/" directory.
./a2 part1

Part 2: For part 2 we had to blend and the apple and the orange image. For this we have used the gaussian function as our kernel function. Doing this we calculate all the gaussian for apple orange and the mask. That is given.

Then we calculate the Laplacian of the of all the images i.e. the orange and the apple images. This we calculate by taking the appropriate gaussian image then taking an upscaled version of the next gaussian and getting their difference.

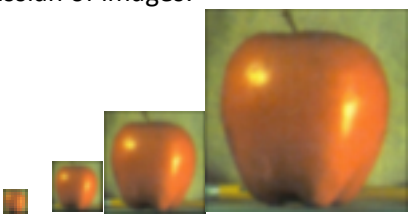
Finally, we use the blending formula $LB_i(x, y) = M_i(x, y) * L1_i(x, y) + (1 - M_i(x, y)) * L2_i(x; y)$; to blend the two images.

Our result:

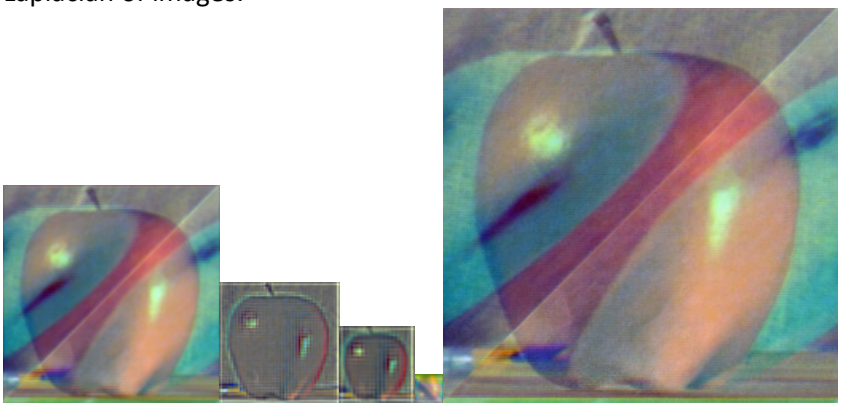


Blended Image

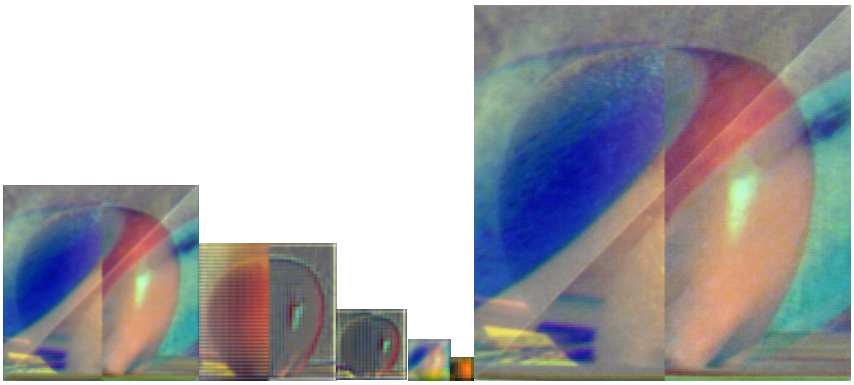
Gaussian of images:



Laplacian of images:



Blended images at every level



Note: images are not to scale.

USAGE : ./a2 part2 apple.jpg orange.jpg mask.jpg

Part 3:

Sift feature matching

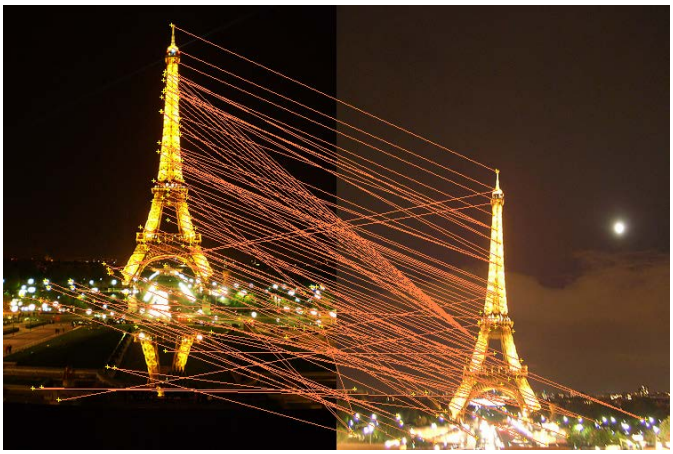
Sub part 1 :-

To compute the matching SIFT descriptors across two images by comparing with the ratio of the Euclidean distance between the closest match and the second-closest match, and comparing to a threshold. For this assignment we have used a threshold for 0.85 for the distance ratio. One more thing to add up here is to stich the images side by side, we initially rescale the images so that the given two images are of equal dimensions before we process them for the feature matching.

The following are the some of the functions that describe our solution to part3 :-

1) stcichImages:- This function rescales the two images, add it beside each other and return the combined image.
2) get_sift_matching:- This function computes the matching sift descriptors and returns the matching pairs as line vector.
3) draw_sift_matches:- This function takes the two images and the matching sift descriptior piars as input and draws the feature_matching image and saves it as a png file.
4) drawLines:- This function takes the image and lineVector as input and draws the appropriate lines and returns the updated images

Resultant image :-



Sub part 2:-

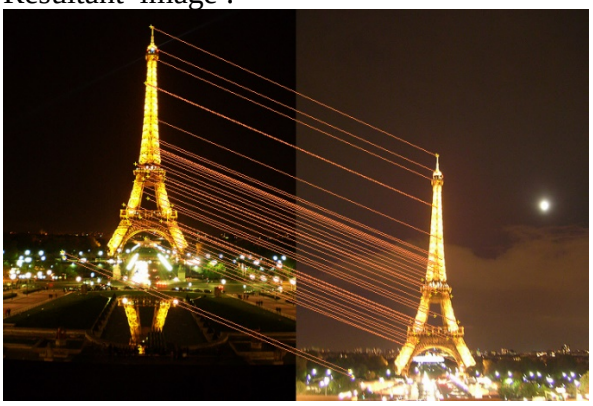
To remove matching outliers from the first part and to filter out the accurate matching descriptors using Ransac algorithm. We have fixed our number of iterations to **30,000** and error threshold to **10**.

The following are the some of the functions that describe our solution to this sub part:-

1) sift_match_pruning :- This function takes two images as input and computes the best matching descriptors between them
2) generateRansacModel :- This function implements the RANSAC algorithm and draws the descriptor matching image.

Apart from the above functions the solution for this sub part also uses the above mentioned functions in sub part 1

Resultant image :-



Usage :-

./a2 part3 (abosolute path of image1) (absolute path of image2)

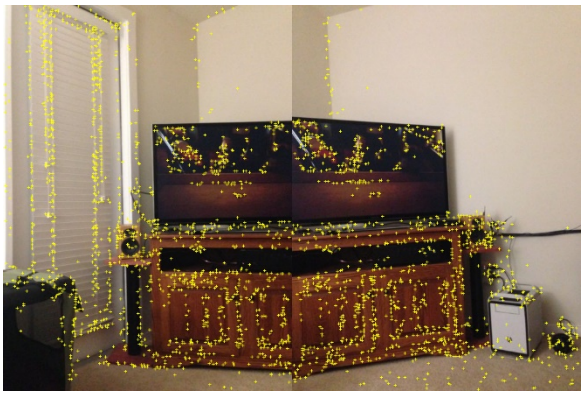
Sample run:-
../a2 part3 images/part3/eiffel_18.jpg images/part3/eiffel_19.jpg

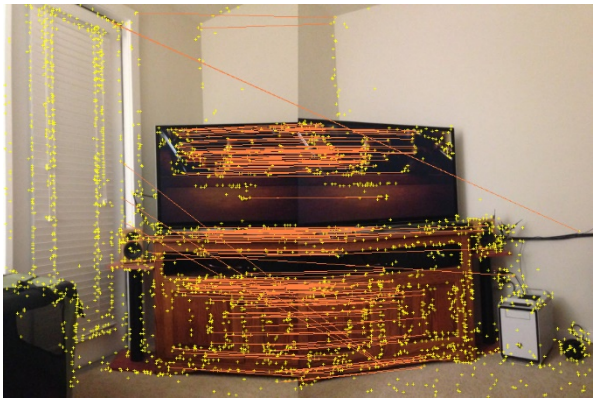
Part 4:

For part 4 as done in previous parts we first found out the sift points on both the images, stitched them together and found out the features points in both the images.

Code snippet for RANSAC
Here we took 4 random points and then calculated the number of inliers for all the points. After this we ran several iterations for them and then selected the points for which we got the maximum inliers.
Snippet:

```
for(j=4;j<int(siftCorrespondence.size());j++)
    {
      x_1 = siftCorrespondence.at(j).x_1;
      y_1 = siftCorrespondence.at(j).y_1;
      x_2 = siftCorrespondence.at(j).x_2;
      y_2 = siftCorrespondence.at(j).y_2;
      X1 = tranformMatrix(0,0)*x_1 + tranformMatrix(0,1)*y_1 + tranformMatrix(0,2)*1;
      Y1 = tranformMatrix(0,3)*x_1 + tranformMatrix(0,4)*y_1 + tranformMatrix(0,5)*1;
      h =  tranformMatrix(0,6)*x_1 + tranformMatrix(0,7)*y_1 + 1;
      X1/=h;
      Y1/=h;
      if(abs(X1-x_2) <= 10 and abs(Y1-y_2) <= 10){
        temp.push_back({x_1,y_1,x_2,y_2});
        inliner++;
      }
    }
```





After this step we found out the homnographies in the images. We used RANSAC for this purpose. Then we used projective transformation for warping our images.

Code Snippet:
For warping

```
cimg_forXY(input_image,i,j)
  {
    float x = (inverse[0][0] * i + inverse[0][1] *j + inverse[0][2]*1);
    float y = (inverse[1][0] * i + inverse[1][1] *j + inverse[1][2]*1);
    float w = (inverse[2][0] * i + inverse[2][1] *j + inverse[2][2]*1);
    if((x/w > 0) && (x/w<input_image.width()) && (y/w>0) && (y/w<input_image.height()))
    {
       output_image(i,j,0,0) = input_image(int(x/w),int(y/w),0,0);
       output_image(i,j,0,1) = input_image(int(x/w),int(y/w),0,1);
       output_image(i,j,0,2) = input_image(int(x/w),int(y/w),0,2);
    }
  }
```

The following were the outputs that we got after warping but unfortunately we couldn't find a way to stitch the images together into a single panorama image.