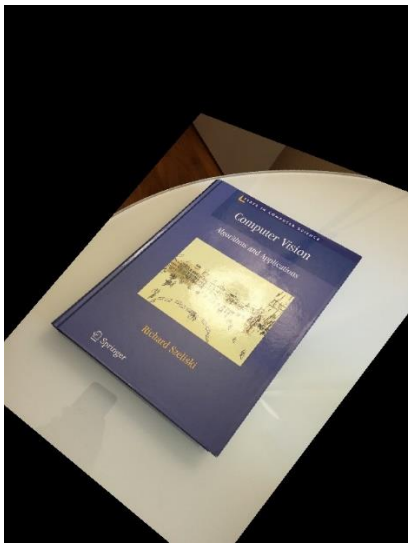
## Part I – Custom Billboards (Image Warping and Homogeneous Matrix)

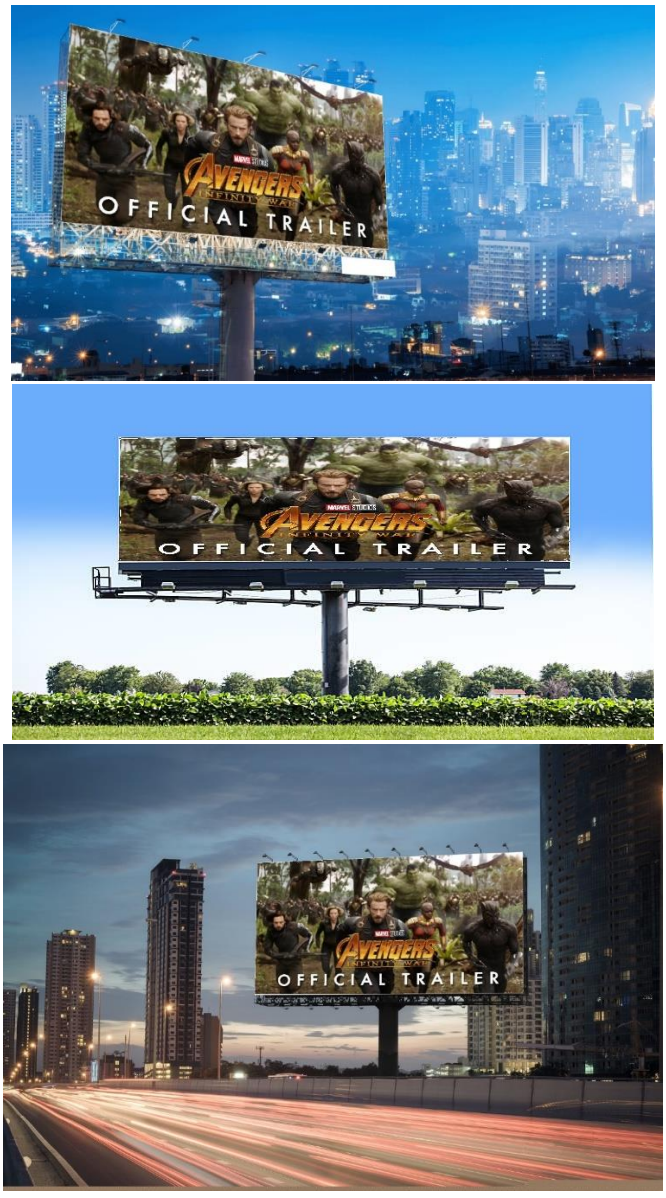
**a)** The given homography matrix is used to transform the original image into the resultant transformed image. As suggested, inverse warping is used to transform the image which results in an image without any gaps in between.
**Output Filename**: lincoln_warped.png

b) Given the four points of matches for two images, a homography matrix is obtained by solving the linear system of equations as suggested in the [1]. And it was applied to the second book image to get it the same coordinate system as book1.jpg to get the image shown below.
**Output Filename**: warped_image2.png




c) Given a poster image, it is projected to the white canvas of the billboard space by identifying the four corner points of the billboard space and creating a homography

matrix with respect to the four corners of the poster image. With the resultant homography matrix, the poster is projected on the different billboard spaces as shown in the images below.




How to run? - **./a2 part1 poster_file.png** will generate the three billboard outputs.

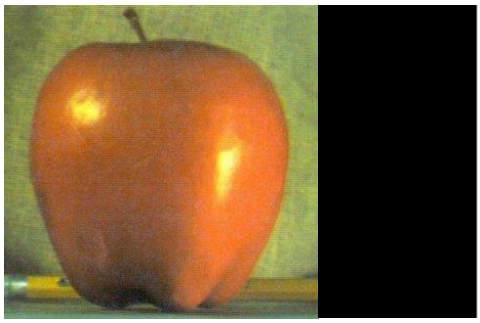
# Part II – Blending

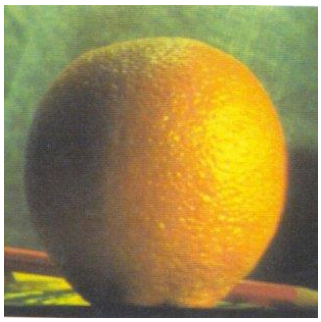
Blending is the process of combining two images together to make a smooth transition between them. The following algorithm was used to seamlessly blend two images using a reference mask.

**Algorithm:**

- Initialized a **Gaussian filter** as follows:

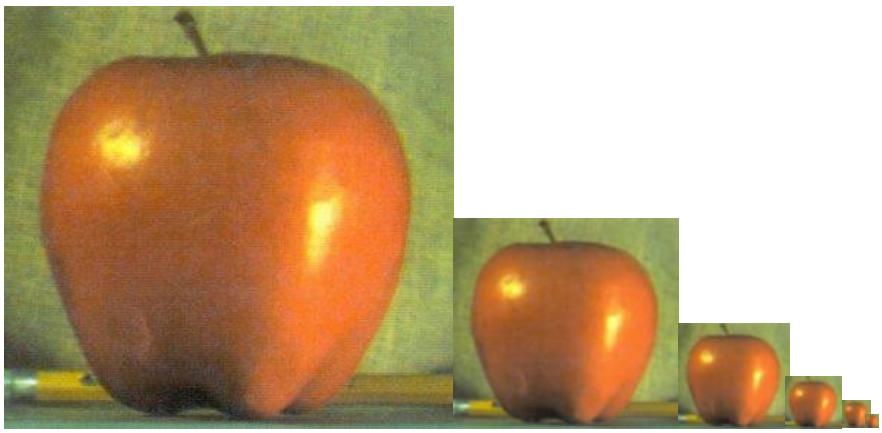
$$Gaussian\ filter, G = \frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

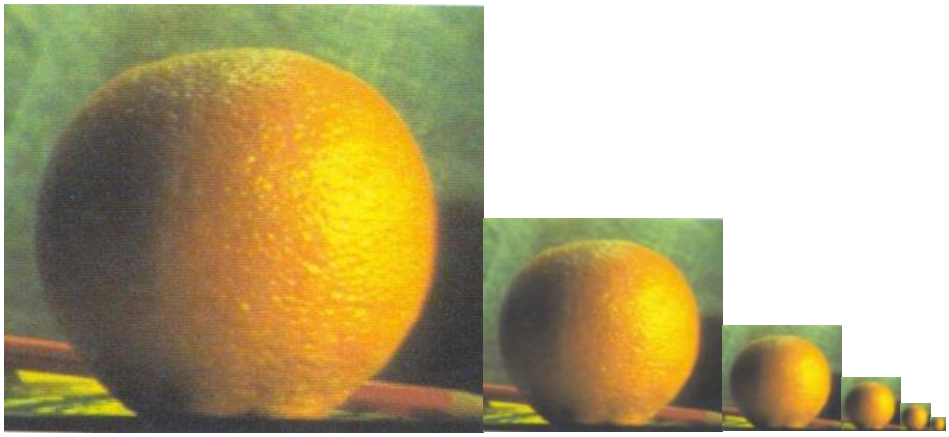




- **Gaussian Pyramid**: A separate pyramid was constructed for each of the two input images and the mask image. Six different levels of the Gaussian pyramid were constructed as follows:

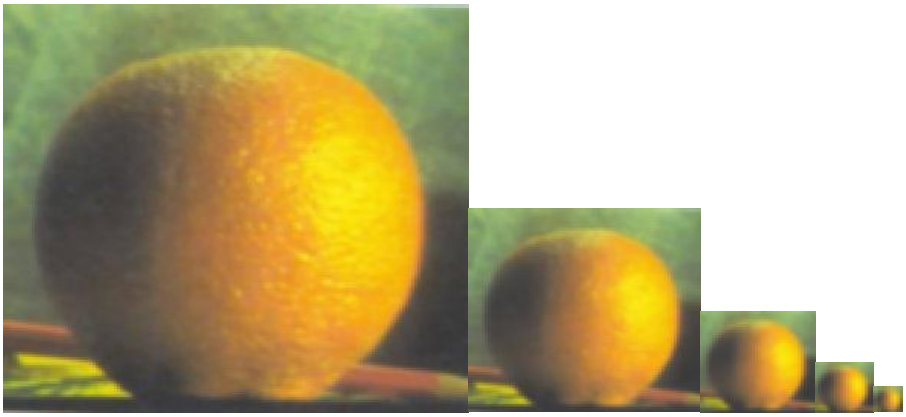
  Gaussian Pyramid Level 0, G0 : Original image (307x307)
  Gaussian Pyramid Level 1, G1 : G0 * G (153x153)
  Gaussian Pyramid Level 2, G2 : G1 * G (76x76)
  Gaussian Pyramid Level 3, G3 : G2 * G (38x38)
  Gaussian Pyramid Level 4, G4 : G3 * G (19x19)
  Gaussian Pyramid Level 5, G5 : G4 * G (9x9)





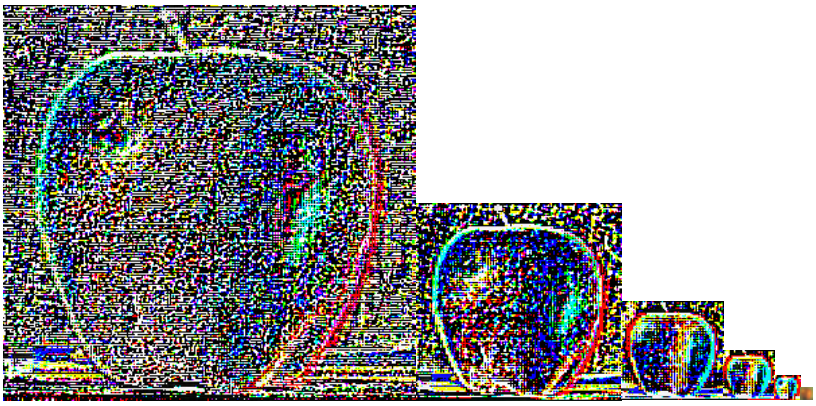
- **Laplacian of Gaussian**: Each level of the Gaussian pyramid was upscaled to the size of the previous layer.

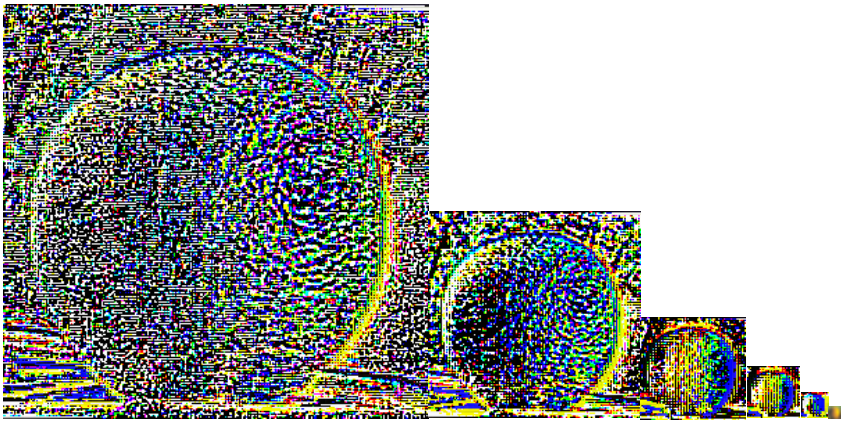
  Laplacian of Gaussian Level 1, LoG1 : Upscale(G1) (307x307)
  Laplacian of Gaussian Level 2, LoG2 : Upscale(G2) (153x153)
  Laplacian of Gaussian Level 3, LoG3 : Upscale(G3) (76x76)
  Laplacian of Gaussian Level 4, LoG4 : Upscale(G4) (38x38)
  Laplacian of Gaussian Level 5, LoG5 : Upscale(G5) (19x19)



- **Laplacian Pyramid**: Laplacian pyramid is constructed by finding the difference between the corresponding level of the Gaussian pyramid and the Laplacian of Gaussian.

  Laplacian Pyramid Level 0, L0 : G0 – LoG1 (307x307)
  Laplacian Pyramid Level 1, L1 : G1 – LoG2 (153x153)
  Laplacian Pyramid Level 2, L2 : G2 – LoG3 (76x76)
  Laplacian Pyramid Level 3, L3 : G3 – LoG4 (38x38)
  Laplacian Pyramid Level 4, L4 : G4 – LoG5 (19x19)
  Laplacian Pyramid Level 5, L5 : G5 (9x9)

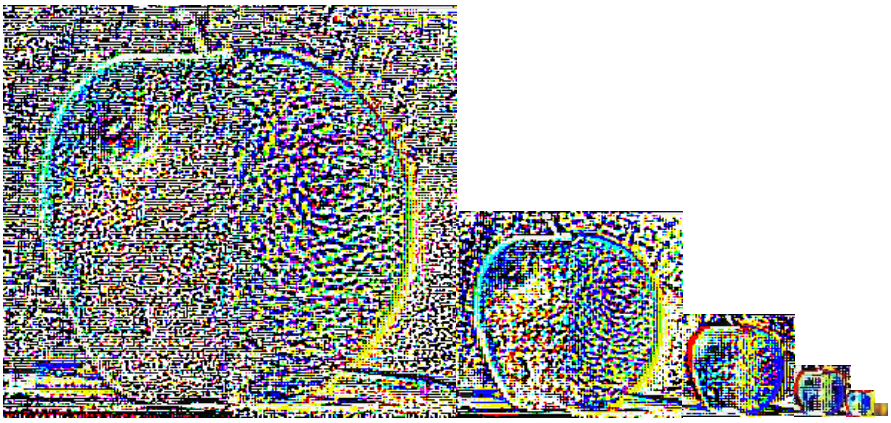




- **Blended Laplacian Pyramid**: Laplacian pyramid is constructed by finding the difference between the corresponding level of the Gaussian pyramid and the Laplacian of Gaussian.

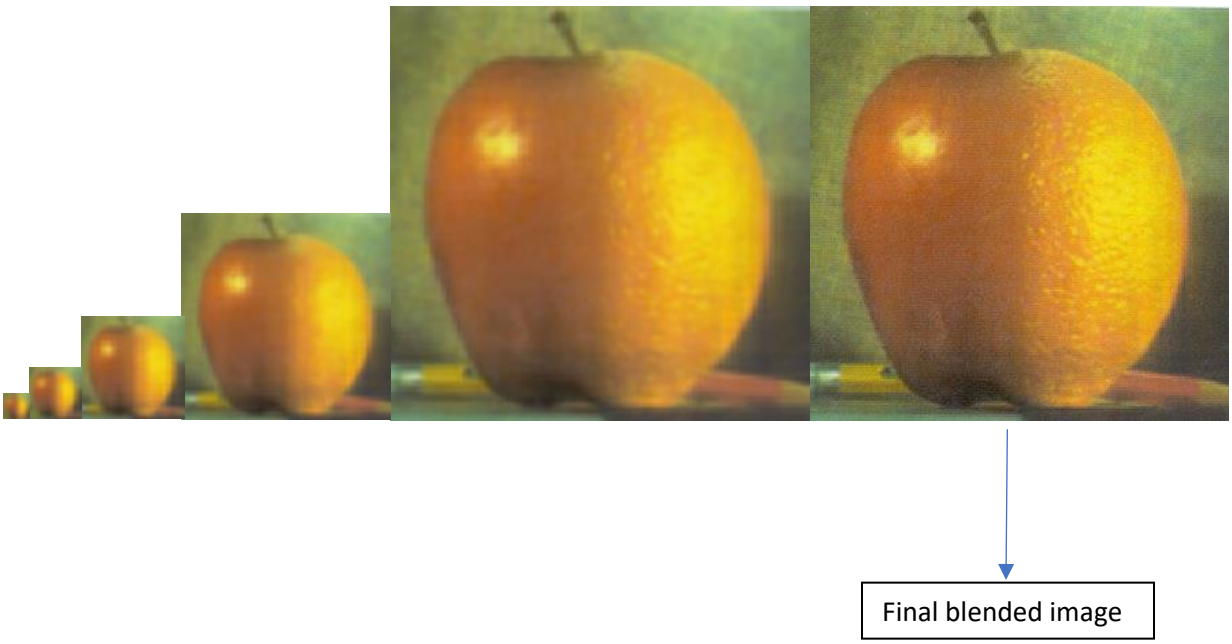
  Blended Laplacian Level 0, BL0 : (G0_mask)L0_Image2 + (255- G0_mask)L0_Image1 (307x307)
  Blended Laplacian Level 1, BL1 : (G1_mask)L1_Image2 + (255- G1_mask)L1_Image1 (153x153)
  Blended Laplacian Level 2, BL2 : (G2_mask)L2_Image2 + (255- G2_mask)L2_Image1 (76x76)
  Blended Laplacian Level 3, BL3 : (G3_mask)L3_Image2 + (255- G3_mask)L3_Image1 (38x38)

Blended Laplacian Level 4, BL4 : (G4_mask)L4_Image2 + (255- G4_mask)L4_Image1 (19x19)
Blended Laplacian Level 5, BL5 : (G5_mask)L5_Image2 + (255- G5_mask)L5_Image1 (9x9)



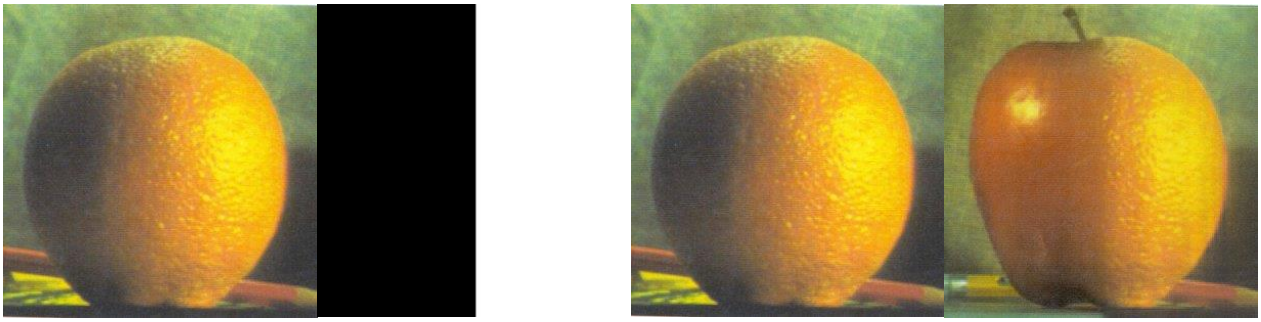
- **Image Reconstruction**: The image is reconstructed by upsizing the smallest of the blended Laplacian to the size of the previous level and convolving it with a Gaussian Kernel, G_new(with 4 times the magnitude as G). This is recursively done on the addition of result of lower layer with the Blended Laplacian of the current layer. R0 is the final blended output.

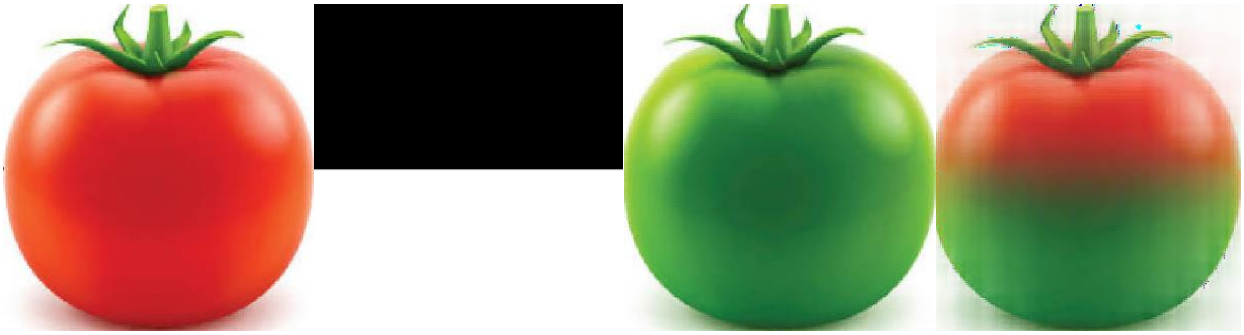
  Reconstructed Image Level 5, R5 : Upscale(BL5) * G_new (9x9)
  Reconstructed Image Level 4, R4 : Upscale(BL4+R5) * G_new (19x19)
  Reconstructed Image Level 3, R3 : Upscale(BL3+R4) * G_new (38x38)
  Reconstructed Image Level 2, R2 : Upscale(BL2+R3) * G_new (76x76)
  Reconstructed Image Level 1, R1 : Upscale(BL1+R2) * G_new (153x153)
  Reconstructed Image Level 0, R0 : Upscale(BL0+R1) (307x307)



**Other Experiments and Outputs:**

Given below are the experiments run and their corresponding outputs. The images are ordered as : input image 1, mask image, input image 2 and the output blended image. We see a perfectly blended output for the apple-orange example and the tomatoes. For the last image, we saw slight noise near the red moon in the blended output. It might be due to the noise in the image because the resolution was not great for the input images.





**To execute the code:**

- Use the following command to run the code for part 2, *./a2 part2 apple.jpg orange.jpg mask.jpg*

  Where apple is the input image 1, orange is the input image 2 and mask is the mask image used.

- Ensure that all the three images are in the same folder as the code while executing
- The blended output will be saved in *Blended_Output.png*

# Part III – Image Matching and RANSAC

In this part, we are asked to perform Feature Matching by using SIFT (Scale Invariant Feature Transform) to detect key points and computing their corresponding descriptors for two images. Using these descriptors, we find the matches through Euclidean distance method. We calculate the distance from one descriptor in image 1 to every other descriptor in image 2. The ratio of the smallest and second smallest distance is taken to determine the match. We have also set a threshold value of 0.8 to select the closest match. The figure below shows all the possible matches between the two images of Eiffel Tower. The lines join one feature from image 1 to its corresponding feature in image 2. We store these matches in two vectors.

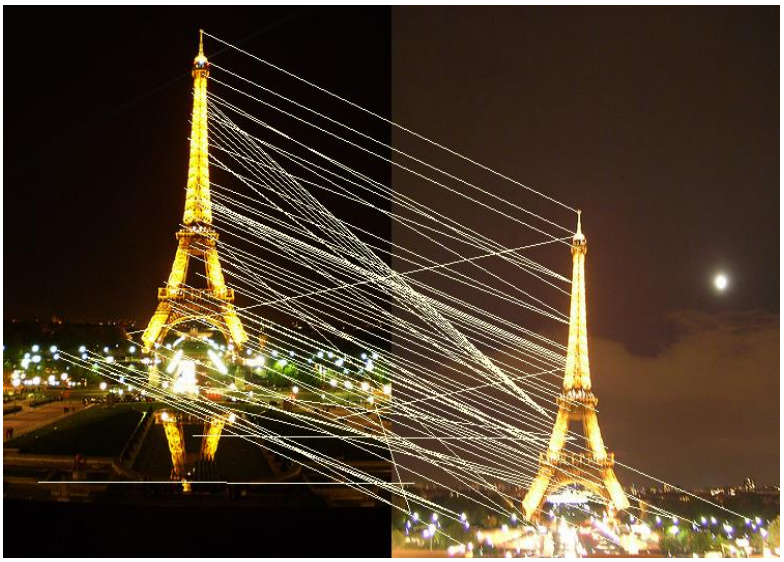


Image 1

**Feature Matching by RANSAC**

We then select 4 feature points randomly from image 1 and their corresponding feature points from image 2. We pass two vectors that contains the descriptors of the matched features to compute the homography matrix. This homography matrix along with the descriptors of the first image is then passed to the transformation function to find the corresponding key points in the second image. To check which set of matched features give maximum accuracy we use SSD. We set a threshold value of 5 to determine the accuracy and to differentiate between the inlier and outlier points. If the value is below threshold then we set it as an inlier. We iterate our random generator 500 times to get the best set of feature matches. The following image is produced after we perform feature matching through RANSAC.

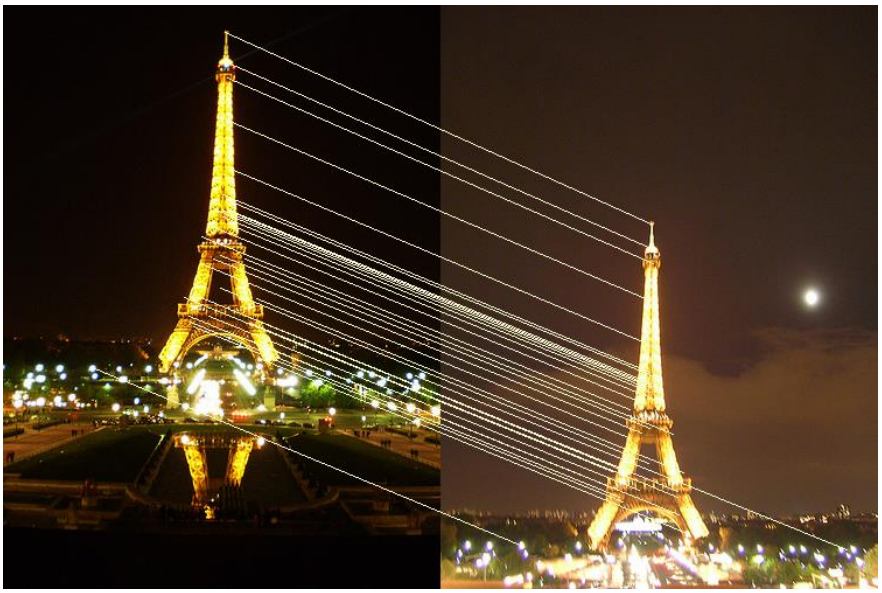


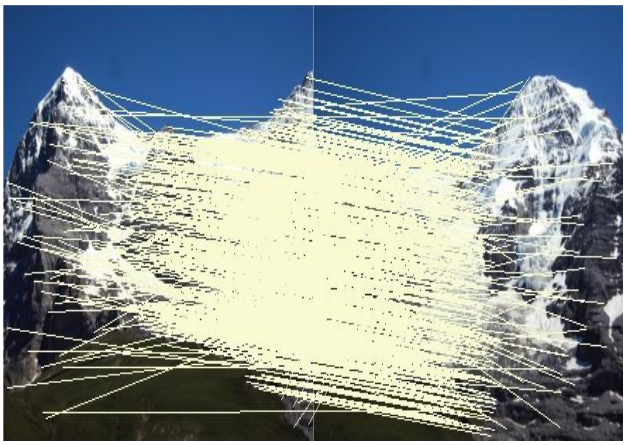
Image 2

By Implementing Feature Matching using both distance and RANSAC method, we find that RANSAC gives better matches than the former. Also, implementing RANSAC is less expensive than implementing the same through distance method. This mostly because we apply brute force method of calculating Euclidean distance between each descriptor.

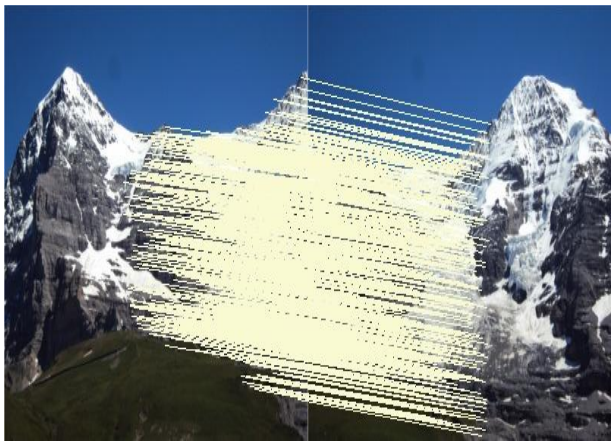
**Experiments and Results:**

Here are few other results of feature matching:

Feature Matching using Brute Force Method



Feature Matching using RANSAC



Our algorithm did not work for the following pair of images:

Feature Matching using Brute Force Method

Feature Matching using RANSAC

**To execute the code:**

- We call our program in the following way: *./a2 part2 eiffel_18.png eiffel_19.png*

# Part IV – Creating Multi-Image Panoramas

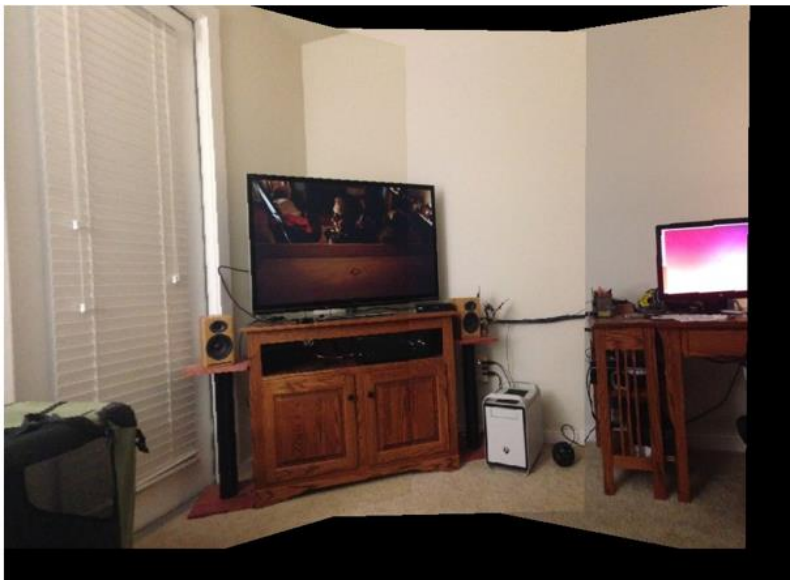
Image stitching was done to combine multiple images (First 3 images in this case) in such a way to form a panoramic image. Image transformations and image stitching techniques were used to arrive at the final panoramic image. Consider the images from left to right as image1, image2, image3.

**Algorithm:**

- Image 3 was transformed to the same coordinate system as image 2. To do this, the best homography matrix was determined through several iterations
- Once the images were transformed, they were mapped onto a bigger canvas with twice the height and thrice the width of the individual images
- Then transformed image 3 was first projected onto the right side of the blank canvas. Following this, the image 2 was projected onto the canvas by transforming it with respect to the current canvas
- Once these two images were projected, image 1 was transformed with respect to this canvas containing images 2 and 3
- Median filter was applied to smoothen the sharp transition between the stitched images. Image blending can be done for seamless transitions. To do this, the mask can be determined by identifying the boundary region.

**Experiments and Outputs:**

The outputs given below, show the result of panoramic image stitching. The images have been perfectly overlaid to create a continuous canvas. In both these case 3 images were combined to create the panorama. Please note that the unwanted empty canvas regions have been cropped for clarity.



However, the algorithm did not work perfectly well in certain other cases. In the output given below, image 3 has been projected accurately but the images 1 and 2

**To execute the code:**

- Use the following command to run the code for part 4,
  *./a2 part4 image1.jpg image2.jpg image3.jpg*
- Ensure that all the three images are in the same folder as the code while executing
- The final output will be saved in *warped_final.png*

REFERENCES:

[1] http://www.csc.kth.se/~perrose/files/pose-init-model/node17.html for the linear equation solving for