# ASSIGNMENT 2: Computer Vision 792

## 1 Question 1

`q1.py` performs a series of similarity transformations on an image. The key steps and functionality are as follows:

- The image is loaded and converted to a NumPy array.
- Four lists of scaling factors ($s$), translation values ($tx$ and $ty$) and dividing values ($d$) are defined.
- The code iterates over all combinations of these values to create a series of similarity transformation matrices using the following formula:

$$H = \begin{bmatrix} s\cos(\pi/d) & -s\sin(\pi/d) & tx \\ s\sin(\pi/d) & s\cos(\pi/d) & ty \\ 0 & 0 & 1 \end{bmatrix}$$

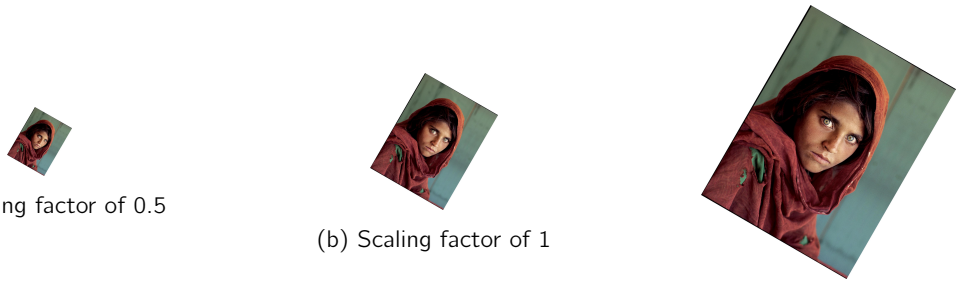
- For each transformation matrix $H$, the `applyhomography` function is applied to the image to produce a transformed image.

(a) Scaling factor of 0.5 (b) Scaling factor of 1 (c) Scaling factor of 2

Figure 1: Rotation by $\pi/3$ with translations of 1 unit in both the $x$ and $y$ directions

(a) Scaling factor of 0.5 (b) Scaling factor of 1 (c) Scaling factor of 2

Figure 2: Rotation by $\pi/4$ with translations of 1 unit in both the $x$ and $y$ directions



(a) Scaling factor of 0.5 (b) Scaling factor of 1 (c) Scaling factor of 2

Figure 3: Rotation by $\pi/6$ with translations of 1 unit in both the $x$ and $y$ directions

(a) 0.5 translation in x direction

(b) 1 translation in x direction

(c) 2 translation in x direction

Figure 4: Rotation by $\pi/4$, scaling factor of 2 and translation of 1 unit in the $y$ direction

### 1.1 Interpretation

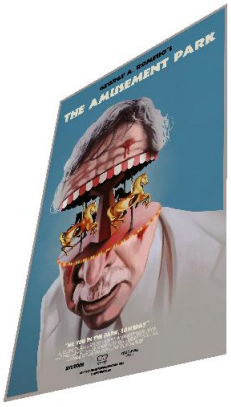
The `applyhomography.py` function is designed to compensate for the shift in the output image's origin. The code transforms the four corners of the input image using the homography matrix H. It then finds the minimum and maximum x and y coordinates. The size of the output image is then set to encompass all these points. This is why we do not see an impact of the translations on the images.

## 2 Question 2

`q2.py` performs a homography transformation on an image and pastes it onto another image. Below is a detailed breakdown:

- **apply_homography Function:**
  - This function takes an image `A` and a homography matrix `H`, then applies the homography transformation.
  - The image's corners are transformed using the homography matrix to determine the size of the transformed output.
  - The inverse homography matrix is used to map each pixel in the output image back to the original image for interpolation.
  - Pixels are interpolated using bilinear interpolation, and pixels outside the transformed region are set to white.
- **find_homography Function:**
  - This function computes the homography matrix `H` using four pairs of corresponding points from the source and destination images.
  - It sets up a system of linear equations and solves it using Singular Value Decomposition (SVD) to find the homography matrix.
- 
  - The code loads a poster image and a building image.
  - It defines source points from the poster and corresponding destination points on the building image.
  - The homography matrix is calculated using these points.
  - The poster is transformed using the homography matrix, and the resulting image is then pasted onto the building image.
  - A mask is created to ensure only non-white pixels of the transformed image are pasted.


(a) Original Movie Poster



(b) Transformed Movie Poster

(c) Building with Movie Poster

Figure 5: Pasting a transformed movie poster onto a building

## 2.1 Interpretation

The transformed poster image contains a white region where there is no image data because the transformation maps some parts of the image to areas that fall outside the original image bounds. To handle this, I created a mask: When pasting the transformed poster onto the building, the mask ensures that only the parts of the poster with actual content are pasted.

After applying the transformation, the origin of the poster shifts to a different position. This happens because the transformation can move the image in ways that push it outside the original image boundries. I compensate for this by calculating `minx` and `miny`. These represent how much the image has shifted in the negative direction from the origin `(0,0)`. In this case, the minimum x and minimum y co-ordinates shift by 35 and 124 units respectively. This offset ensures that the transformed poster aligns correctly with the intended location on the building image.

## 2.2 Experimentation

The original image is of size 1400 x 2000 and the building image is of size 528 x 704.

### 2.2.1 Case 1 : Large Input Image

I mantain the original poster size as it is already quite large. As one can observe, there are aliasing artifacts such as jagged edges and loss of fine detail. The downscaling process during the transformation fails to capture the high frequency content.

### 2.2.2 Case 2 : Input Image of Same Size

I changed the input image to a size of 528 x 704 to resemble the size of the output image. As one can observe, the image suffers from minimal aliasing.

### 2.2.3 Case 3 : Small Input Image

The input image was now 250 x 350. The output poster is more blurred, but there is not as much aliasing artifacts as there were when the image was larger.

(a) (b)

Figure 6: Building with 1400x2000 Movie Poster

(a) (b)

Figure 7: Building with 528x704 Movie Poster

(a) (b)

Figure 8: Building with 250x250 Movie Poster

# 3 Question 3

`q3.py` changes the perspective we view the 3D art in `bricks.jpg` so that we view it as if we were viewing it from the top. The `find_homography` function computes the homography matrix used to transform points from a source image to a destination image.

- **Matrix Construction:** The function creates a matrix `A` from the source and destination points. Each pair of corresponding points generates two rows of `A`, based on specific linear equations.
- **Singular Value Decomposition (SVD):** The matrix `A` is decomposed using SVD. The last row of the resulting matrix `Vh` provides the components needed to calculate the homography matrix.
- **Homography Matrix:** The final homography matrix is obtained from the last row of `Vh`, normalized by its last element. This matrix is reshaped into a $3 \times 3$ format.

## Applying the Homography

The code then performs the following steps:

- **Read and Convert Image:** The input image is read using OpenCV and converted to the RGB color space.
- **Define Points:** Source and destination points are defined for the homography transformation.
- **Compute Homography:** The homography matrix is computed using the previously defined function.
- **Apply Homography:** The image is transformed using the computed homography matrix.
- **Save Output:** The transformed image is saved after converting it back to the BGR color space.


(a) Original Image (b) Perspective of 3D Art from above

## 3.1 Interpretation

Given the source co-ordinates of the trapezium which are:
`[[344,499],[30,721],[847,531],[789,824]]`
and the destination co-ordinates of the square region we want to transform it into:
`[[0, 0], [0, 900], [900, 0], [900, 900]]` When I applied the homography, the trapezium image is warped to fit into the square (as the tile covered by the artwork is square).