Source code

Important Notice:

To keep the size of the document low, the following code only includes the functions that I implemented and the point measurements. I have included my complete (non-cleaned up) python notebook with all outputs in my submission as well for your convenience.

```
import numpy as np
import cv2
import math
import matplotlib.pyplot as plt

class Point:
    def __init__(self, x, y):
        """Defines a point using its physical space coordinates"""
        self.x = x
        self.y = y
        self.hc = self.get_hc()
    @classmethod
    def from_hc(cls, hc):
        """Defines a point from its representation in homogeneous
coordinates"""
        if np.isclose(hc[2],0):
            x = hc[0]
            y = hc[1]
        else:
            x = hc[0] / hc[2]
            y = hc[1] / hc[2]
        return cls(x, y)
    def get_hc(self):
        """Returns the point in homogeneous coordinates"""
        return np.array([self.x, self.y, 1])
    def __repr__(self):
        """To string method for debugging"""
        return f"Point(x={self.x}, y={self.y}, hc={self.hc})"

class Homography():
    def __init__(self):
        pass
    def get_first_six_cols(self, x_points):
        zero_vec = np.zeros(3)
        eqn1_first6cols = np.vstack([np.hstack((point.get_hc(),
zero_vec)) for point in x_points])
        eqn2_first6cols = np.vstack([np.hstack((zero_vec,
point.get_hc())) for point in x_points])
        return np.vstack([[eqn1, eqn2] for eqn1, eqn2 in
zip(eqn1_first6cols, eqn2_first6cols)]) # Stack the rows in an
```

```
interleaved fashion

    def estimate_projective_homography(self, x_points,
x_prime_points):
        # Lets build the first 6 columns of the matrix we are
interested in:
        first6cols = self.get_first_six_cols(x_points)

        # Now we only need the last 2 columns
        # x' is in the shape: x1' y1' 1, x2' y2' 1 etc
        # x is: x1 y1 1, x2 y2 1, ...
        # So we take the outer product of the first two elements in
the homogeneous form to get the remaining portion:
        last2cols_list = []
        for x, x_prime in zip(x_prime_points, x_points):
            last2cols_list.append(-np.outer(x.get_hc()[:2],
x_prime.get_hc()[:2]))
        last2cols = np.vstack(last2cols_list)
        full_matrix = np.hstack((first6cols, last2cols))

        x_prime_matrix = np.vstack([np.vstack((x_prime.get_hc()[0],
x_prime.get_hc()[1])) for x_prime in x_prime_points])
        H = np.matmul(np.linalg.inv(full_matrix), x_prime_matrix)
        H = np.vstack((H, np.array(1))).reshape((3,3))
        return H

    def estimate_affine_homography(self, x_points, x_prime_points):
        # ONly 3 points are needed
        x_points = x_points[:3]
        x_prime_points = x_prime_points[:3]

        # Get first 6 columns similar to the general projective
homography
        # However, in the case of an affine matrix, we only need those
6 columns
        full_matrix = self.get_first_six_cols(x_points)

        # Create x' matrix: [x_1', y_1', x_2', y_2', ...]
        x_prime_matrix = np.vstack([np.vstack((x_prime.get_hc()[0],
x_prime.get_hc()[1])) for x_prime in x_prime_points])

        H = np.matmul(np.linalg.inv(full_matrix), x_prime_matrix)
        last_row = np.array((0, 0, 1)).reshape(-1, 1)
        H = np.vstack((H, last_row)).reshape((3,3))
        return H

    # Used for the bonus. Explanations of the code are included in my
report
    def get_normalizing_homography(self, image):
```

```
        h, w = image.shape[:2]
        print(h, w)
        H_norm = [2/w, 0, -1, 0, 2/h, -1, 0, 0, 1]
        H_norm = np.array(H_norm, dtype=np.float32).reshape((3,3))
        return H_norm
    def get_denormalizing_homography(self, image):
        h, w = image.shape[:2]
        H_norm = [w/2, 0, w/2, 0, h/2, h/2, 0, 0, 1]
        H_norm = np.array(H_norm, dtype=np.float32).reshape((3,3))
        return H_norm
    def get_rotational_homography(self, alpha):
        H_rot = [math.cos(alpha), -math.sin(alpha), 0,
math.sin(alpha), math.cos(alpha), 0, 0, 0, 1]
        H_rot = np.array(H_rot, dtype=np.float32).reshape((3,3))
        return H_rot
    def get_horizontal_tilt_homography(self, alpha):
        H = [1,0,0,0,1,0, math.cos(alpha),0,1]
        H = np.array(H, dtype=np.float32).reshape((3,3))
        return H
    def get_vertical_tilt_homography(self, alpha):
        H = [1,0,0,0,1,0,0,math.sin(alpha), 1]
        H = np.array(H, dtype=np.float32).reshape((3,3))
        return H

def reverseMapHomography(source_img, homography, target_size):
    new_image = np.zeros((target_size[0], target_size[1],
source_img.shape[2]), dtype=source_img.dtype)
    H_inv = np.linalg.inv(homography)

    # cv2 is in (y,x) format
    for i in range(target_size[0]):
        for j in range(target_size[1]):
            target_hccoord = np.array((j,i,1))

            # Find the corresponding coordinate from the image I want
to map over
            src_hccoord = np.matmul(H_inv, target_hccoord)
            src_hccoord = (src_hccoord/src_hccoord[2]).astype(int) #
ensure x_3 = 1

            if 0 <= src_hccoord[0] < source_img.shape[1] and 0 <=
src_hccoord[1] < source_img.shape[0]:
                new_image[i,j] = source_img[src_hccoord[1],
src_hccoord[0]]
    return new_image

def mapHomography(source_img, homography, target_size):
    new_image = np.zeros((target_size[0], target_size[1],
source_img.shape[2]), dtype=source_img.dtype)
```

```
    for i in range(source_img.shape[0]):
        for j in range(source_img.shape[1]):
            curr_hc_coord = np.array((j, i, 1))

            new_hc_coord = np.dot(homography, curr_hc_coord)
            new_hc_coord = (new_hc_coord/new_hc_coord[2]).astype(int)
# Ensure x_3 = 1

            # Check x and y bounds:
            if 0 <= new_hc_coord[0] < target_size[1] and 0 <=
new_hc_coord[1] < target_size[0]:
                new_image[new_hc_coord[1]][new_hc_coord[0]] =
source_img[i][j]

    return new_image

def map_image1_onto_roi_on_image2_inrgb(source_img, target_img,
target_roi, homography):
    # This function is used to map the photo of Alex onto the ROI in
the frame-photo
    # Warp the alex image following the previously calculated
homography
    warped_alex = reverseMapHomography(source_img, homography,
target_img.shape)

    # Create a mask of the polygon created by the image frame that I
want to cover
    # I will keep the area outside of the mask
    # Note: cv2 polygons only accept integer pixels:
    polygon_coords = np.array([[int(x), int(y)] for x,y in
target_roi], dtype=np.int32)
    mask = np.zeros_like(target_img, dtype=np.uint8)
    cv2.fillPoly(mask, [polygon_coords], (255, 255, 255))

    # Remove the frame portion from the image
    mask_inv = cv2.bitwise_not(mask)
    img_background = cv2.bitwise_and(target_img, mask_inv)

    # Add the Alex photo in with projective transformation warping in
    final_image = cv2.add(img_background, cv2.bitwise_and(warped_alex,
mask))

    # Convert the image to RGB for matplotlib
    final_rgb_image = cv2.cvtColor(final_image, cv2.COLOR_BGR2RGB)
    return final_rgb_image
```

Point measurements for given images

```
img1_pixel_coords = [(420.590909090909, 841.6818181818182),
(2558.772727272728, 874.409090909091), (2416.954545454546,
2270.7727272727275), (682.409090909091, 3154.4090909090914)]
img2_pixel_coords = [(518.772727272727, 1430.7727272727275),
(1871.5000000000005, 830.7727272727275), (1958.7727272727275,
2805.318181818182), (464.2272727272725, 2674.409090909091)]
img3_pixel_coords = [(1216.9545454545455, 538.4090909090905),
(2973.318181818182, 2305.681818181818), (1806.045454545455,
3178.409090909091), (256.9545454545455, 1814.772727272727)]
alex_pixel_coords = [(126.51136363636363, 201.73484848484838),
(689.6742424242425, 68.59090909090901), (698.6704545454545,
638.9507575757575), (133.70833333333331, 629.9545454545455)]
```

Point measurements for my images

```
myimg1_pixel_coords = [(86.69264069264068, 150.39393939393938),
(555.3073593073593, 141.02164502164499), (524.8474025974026,
766.6222943722944), (121.83874458874459, 745.5346320346321)]
myimg2_pixel_coords = [(194.09090909090907, 23.811688311688158),
(458.24675324675337, 182.77272727272714), (486.29870129870136,
624.590909090909), (135.64935064935065, 603.551948051948)]
myimg3_pixel_coords = [(31.458874458874448, 368.29978354978357),
(603.168831168831, 431.56277056277054), (389.94913419913416,
787.7099567099567), (47.86038961038963, 799.4253246753248)]
myimg4_pixel_coords = [(195.10768398268402, 165.05681818181802),
(426.9799783549784, 37.52705627705609), (426.9799783549784,
777.1996753246754), (222.93235930735938, 619.5265151515152)]
el_cap_pixel_coords = [(76.9193548387097, 114.38709677419342),
(546.5967741935484, 181.48387096774195), (670.4677419354838,
1027.9354838709678), (92.40322580645164, 1089.8709677419356)]
```