Image Processing HW_2

Student -1- : Mostufa Jbareen (212955587) מוסטפא גבארין

Student -2- : Mohammed Egbaria (318710761) מוחמד אגבאריה

Problem 1:

Section a:

Image 1:

Our choice: Histogram Equalization

In given image 1 we want to see the details of the moon, and we almost one dominant color (grey) , so we chose HE because it is the best one in altering the colors in this type of image, it distributes (psu equally) the pixels to different colors in addition to increasing the contrast and the brightness, which allow is to see the difference and details of the moon (we can see the hole and the mountains on the moon in this method while the other two we cannot).

In BCS and Gamma Correction we will not be able to see the details of the moon because both methods do not distribute the pixels as we distribute them in HE. BCS and Gamma Correction does not care about the quantity of the pixels while HE does, which makes HE a better method for showing details in an image.

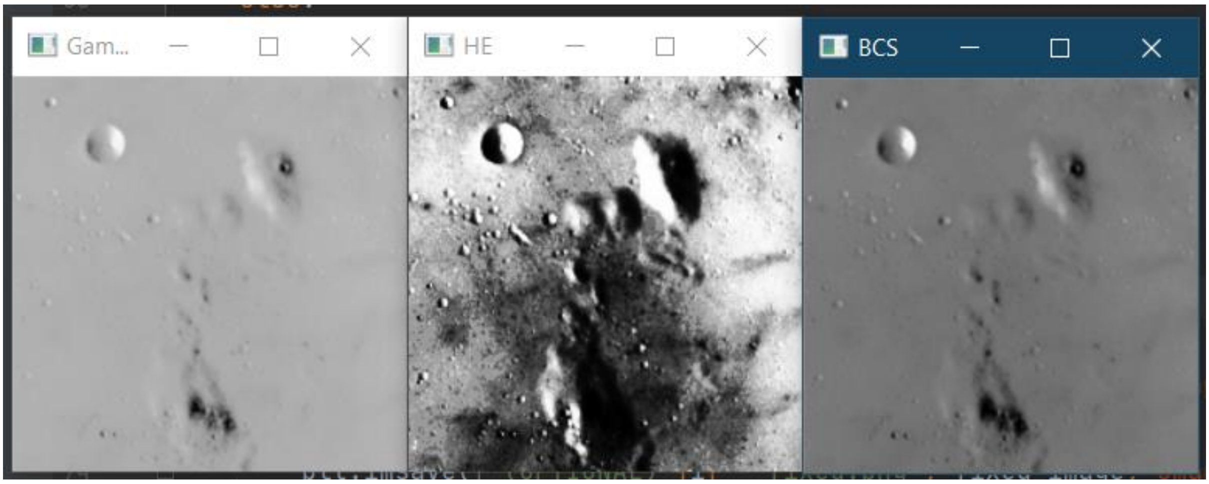


Image 2:

Our choice: Gamma Correction (gamma = 1/2.2)

We can see a lot of dark in given image 2, and we can not see the all the details in the image (we cannot see the house on the right or full structure of the house on the left), so we need a way to brighten the picture in those dark areas. We chose gamma correction with gamma = 1/2.2 (written in the class material that this gamma value is good), because gamma correction with gamma < 1 improves smoothly dark images with non-linear function (smooth transitions and edges).

We did not choose BCE, because the image already has a high contrast (look the street lamp and the dark area), and not HE because, we have a lot of the color black in the picture, so HE tends to take those pixels and color them in more white colors, which results to let the image has a lot of ugly transitions (e.g. white pixels between black pixels)

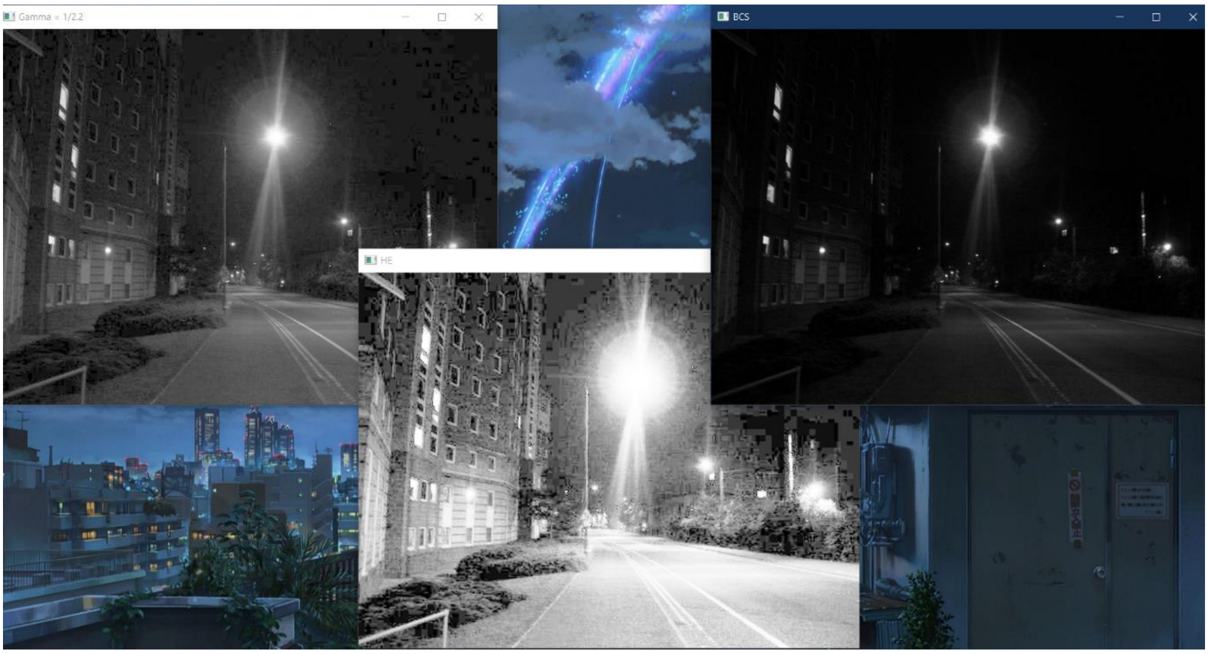
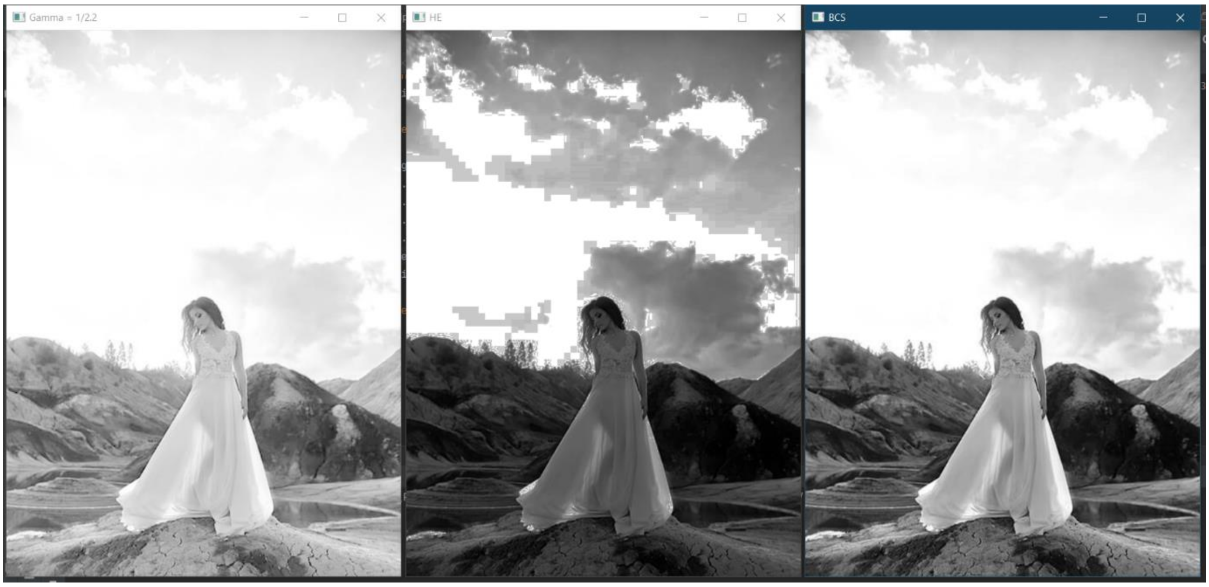


Image 3:

Our choice = Maximum Brightness and Contrast Stretching.

In given image 3, we can see a lot of colors, and the image it self has a high contrast, so we chose Maximum Brightness and Contrast Stretching because, we want to maintain the diversity of the colors.

We did not choose Gamma Correction because gamma tend to brighten dark areas (gamma < 1) or darken bright areas (gamma >1) and we do not want to do that, it is ruining the diversity of the colors, and we did not choose HE because, we can see the image has a lot of the white color, so HE will tend to take some of those pixels and color them in blacker color, which will ruin the white area in the picture (and the sky will be black).



Section b:

Image 1: Histogram Equalization

Image 2: Gamma Correction (with gamma = 1/2.2)

Image 3: Brightness and Contrast Strech (Maximum Brightness Contrast Stech)

Problem 2:

Section a:

Filling matches.txt for each puzzle, example for puzzle_affine_1:

Where (x_1, y_1) *is point in image* 1 *and* $(x_{k=2}, y_{k=2})$*is point in image* $k = 2$.

*C:\CS Courses\Image Processing\Hw\HW2\q2\puzzles\puzzle_affine_1\matches.txt - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

motzamatches.txt | matches.txt | matches.txt

```
## image 2
# first matching
499 159 #img1
217 265 #img2

# second matching
264 94  #img1
256 115 #img2

# third matching
493 416 #img1
16 276  #img2

```

(x_1, y_1) - Im1

(x_2, y_2) - Im2

Section b:

Running on all puzzles, and at the start of each iteration we get the required data on the puzzle (such as matches, is affine and the n images of the puzzle) by calling the function prepare_puzzle.

```
if __name__ == '__main__':
    # puzzle names list
    lst = ['puzzle_affine_1', 'puzzle_affine_2', 'puzzle_homography_1']

    # running on and solving all puzzles
    for puzzle_dir in lst:
        print(f'Starting {puzzle_dir}')

        # preparing paths
        puzzle = os.path.join('puzzles', puzzle_dir)
        pieces_pth = os.path.join(puzzle, 'pieces')
        edited = os.path.join(puzzle, 'abs_pieces')

        # section b
        matches, is_affine, n_images = prepare_puzzle(puzzle)
```

Section c:

Implementing get_transform function, which takes the matching points between two images (in our case image 1 and image k), and takes the type of the transform (is affine or not (which means perspective in our case)), and return the transform from image 1 to image k (which is a matrix).

```
# section c
def get_transform(matches, is_affine):
    """
    calculating the transform from img-1 to img-k
    :param matches: matches is of (3|4 X 2 X 2) size. Each row is a match - pair of (kp1,kp2) where kpi = (x,y)
    :param is_affine: the type of the transformation
    :return: t - the transformation from img-1 to img-k
    """
    # separating matches data to dst_point = image-1 points, and src_points = image-k points
    src_points, dst_points = matches[:, 0], matches[:, 1]
    src_points = src_points.astype(np.float32)
    dst_points = dst_points.astype(np.float32)

    # checking the type of the transformation, and we react according to it and returning the transformation
    if is_affine:
        t = cv2.getAffineTransform(src_points, dst_points)
    else:
        t = cv2.getPerspectiveTransform(src_points, dst_points)
    return t
```

Section d:

Implementing inverse_transform_target_image function, which takes image k as target_img, the transform from image 1 to image k as original_transform, and the required output image size.
And returns the image k with the required size and in it's place in the puzzle, by applying the warping and inverse transforming accordingly to the type of the transform, as it shows below in the picture.

```
# section d
def inverse_transform_target_image(target_img, original_transform, output_size):
    """
    calculating and returning the transformed target_image (in it's right place in the puzzle) (target_image_absolute),
    using cv2.warpPerspective() if the transform is perspective or if cv2.warpAffine() the transform is affine
    :param target_img: image-k
    :param original_transform: the transform from image-1 to image-k
    :param output_size: (image 1 width image 1h height)
    :return: image-k_absolute (in it's right place in the puzzle)
    """

    # checking if the transform is affine or perspective by checking original transform shape (3,3) | (2,3)
    if original_transform.shape == (3, 3):
        return cv2.warpPerspective(target_img, numpy.linalg.inv(transform), (output_size[1], output_size[0]),
                                   flags=cv2.INTER_LINEAR)
    else:
        return cv2.warpAffine(target_img, cv2.invertAffineTransform(transform), (output_size[1], output_size[0]),
                              flags=cv2.INTER_LINEAR)

```

Section e:

Implementing stitch function, which takes two images and stitch them together, by creating avg(image1, image2), and then fixing the places where imag1 and imag2 do not overlap (the places where image 1 is valid and image 2 is black background = 0 and vice versa, so those places or $\frac{x_1+0}{2}$ or $\frac{x_2+0}{2}$) by adding the 0.5 * difference between image 1 and image 2.

```
# section e
def stitch(img1, img2):
    """
    averaging img1, and img2 using cv2.addWeighted(img1, 0.5, img2, 0.5, 0), then adding
    0.5 * (where img1 and img2 do not overlap) by
    cv2.addWeighted(cv2.addWeighted(img1, 0.5, img2, 0.5, 0), 1, cv2.absdiff(img1, img2), 0.5, 0)
    and that is because when we averaged on img1 and img2, we also averaged on the places they do not overlap,
    which mean  img1 = 0.5 * img1 and img2 = 0.5 * img2 in those places, so we add them back
    :param img1: current puzzle progress
    :param img2: new image we add stitch to puzzle
    :return: puzzle after stitching imag2
    """
    return cv2.addWeighted(cv2.addWeighted(img1, 0.5, img2, 0.5, 0), 1, cv2.absdiff(img1, img2), 0.5, 0)

```

Section f:

After preparing the puzzle, we run on all images and calculate the transform from image 1 to image k by using get_transform(), and then we get the piece k absolute using inverse_transform…(), and we stitch it to our current solution, at the end of the loop we have the solution strored in final_puzzle which we are going to save after, and while looping we saved the pieces_absolute in the right directory.

```
        matches, is_affine, n_images = prepare_puzzle(puzzle)

        # reading 1'st piece, which is placed correctly
        piece_1_path = os.path.join(pieces_pth, "piece_1.jpg")
        piece_1 = cv2.imread(piece_1_path)

        # saving piece_1_absolute, which is given to us.
        cv2.imwrite(os.path.join(edited, f"piece_1_absolute.jpg"), piece_1)

        # final_puzzle = the solved puzzle
        # we initialize the final_puzzle to be the piece_1, and we stitch the other pieces to it.
        final_puzzle = piece_1

        # running on all puzzle pieces except the first one because it is already placed right.
        for k in range(2, n_images + 1):
            # calculating the transformation from piece_1 to piece_k
            transform = get_transform(matches[k - 2, :, :], is_affine)

            # reading the piece_k
            piece_k_path = os.path.join(pieces_pth, f"piece_{k}.jpg")
            piece_k = cv2.imread(piece_k_path)

            # calculating piece_k_absolute using inverse_transform_target_image function
            im_k_absolute = inverse_transform_target_image(piece_k, transform, piece_1.shape)

            # saving piece_k_absolute in the abs_pieces folder
            cv2.imwrite(os.path.join(edited, f"piece_{k}_absolute.jpg"), im_k_absolute)

            # stitching piece_k_absolute to our current puzzle progress using stitch function
            final_puzzle = stitch(final_puzzle, im_k_absolute)
```

And after that we plot the solution, and we save the final puzzle which is contain the puzzle solution in the right directory, and continuing in our loop on the next puzzle.

```
        # section f
        # outputting the solution
        cv2.imshow("Solution of" + puzzle, final_puzzle)
        cv2.waitKey(0)

        # saving puzzle solution in the right folder
        sol_file = f'solution.jpg'
        cv2.imwrite(os.path.join(puzzle, sol_file), final_puzzle)
```