תרגיל בית 1 עיבוד תמונה

מגישים:

אור דינר - 207035809

איתמר- 207931296

שאלה 1

סעיף א': **בחירת תיקון לכל תמונה.**

עבור תמונה 1: התיקון הנדרש הוא שיווי היסטוגרמה, מכיוון שגווני התמונה מאוד דומים אחד לשני ולעין האנושית קשה להבחין איך הגוונים משתנים בעצמה. בעזרת שיווי היסטוגרמה נוכל לתת לכל גוון גוון חדש שיאפשר לעין האנושית להבחין טוב יותר בשינוי הגוונים.

עבור תמונה 2: התיקון הנדרש הוא הוא תיקון גמא, מכיוון שהתמונה יחסית חשוכה ובכך לעין האנושית קשה לראות חלק מהפרטים תיקון הגמא יגרום לתמונה להפוך להיות בהירה יותר אבל בצורה מדורגת שתאפשר לעין האנושית להבחין.

עבור תמונה 3: אף תיקון מהשלושה שבשאלה לא ישפר את התמונה ממש מכיוון שכשצילמו את התמונה החלק של השמיים "נשרף" ולאיזור גדול בתמונה יש ערך פיקסלים זהה ושווה ל-255 הערך המקסימלי. כל פעולה מהשלוש שתעשה לא תוכל לגרום לקבלת ערכים שונים לפיקסלים אלו.
אם זאת ניתן טיפה להוריד את הניגודיות ולהעלות טיפה את הבהירות על מנת לקבל תמונה פחות בוהקת.

סעיף ב': **מימוש התיקונים.**

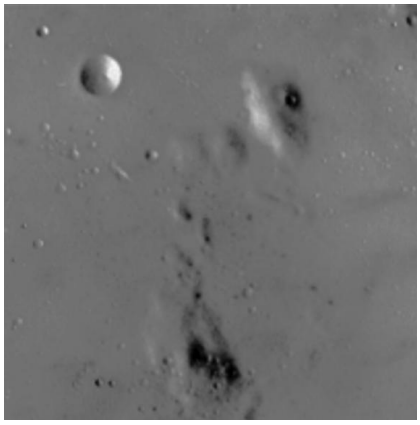
מימשנו פונקציה לכל סוג תיקון ועל ידי פונקציית 'apply_fix' אנחנו מפעילים את הפונקציה הנדרשת לכל תמונה.

```
def apply_fix(image, id):  1 usage
    # Your code goes here
    if id == 1:
        return histogram_equalization(image)
    elif id == 2:
        return gamma_correction(image, gamma=1.4)
    elif id == 3:
        return brightness_contrast_stretching(image, alpha=0.9, beta=15)
    else:
        return image
```

עבור תמונה 1:

שיווי היסטוגרמה

$\longrightarrow$



עבור תמונה 2:

תיקון גמא

$\gamma = 1.4$

$\longrightarrow$

עבור תמונה 3:

תיקון בהירות וניגודיות

ניגודיות = 0.9

בהירות = 15

$\longrightarrow$

שאלה 2

סעיף א': **לקיחת נקודות תואמות.**

עבור שני הפאזלים הראשונים האפינים לקחנו שלוש נקודות תואמות בין החתיכה הראשונה לכל חתיכה אחרת. עבור הפאזל ההומוגרפי לקחנו 4 נקודות משותפות בין החתיכה הראשונה לאחרות ורשמנו הכל בקבצי הטקסט.

סעיף ב': **הרצת 'prepare_puzzle'**

הרצנו את prepare_puzzle וקיבלנו את הנקודות התואמות, האם החתיכות אפיניות ואת מספר התמונות.

סעיף ג': **מימוש 'get_transform'**

בפונקציה הזאת אנו מקבלים את הנקודות התואמות בין הפאזלים והאם החתיכות אפיניות אחת לשניה או לא. במקרה שהחתיכות אפיניות נקבל את הטרנספורמציה הדרושה על ידי הפונקציה הספרייה 'cv2.estimateAffine2D'. במקרה שהחתיכות לא אפיניות נקבל את הטרנספורמציה על ידי 'cv2.findHomography'.

```python
def inverse_transform_target_image(target_img, original_transform, output_size):  1 usage
    """
    Perform the inverse transform to bring the xth image into the 1st image's canvas.

    Args:
        target_img (numpy.ndarray): The xth image.
        original_transform (numpy.ndarray): The transformation matrix from the 1st to xth image.
        output_size (tuple): The output canvas size (width, height).

    Returns:
        numpy.ndarray: The transformed image.
    """
    if original_transform.shape == (2, 3):  # Affine transformation
        # Convert 2x3 affine to 3x3 by adding [0, 0, 1]
        affine_transform = np.vstack([original_transform, [0, 0, 1]])
        # Invert the 3x3 matrix
        inverse_transform = np.linalg.inv(affine_transform)[:2, :]  # Take first 2 rows back as 2x3
        # Use cv2.warpAffine for affine transformations
        transformed_image = cv2.warpAffine(target_img, inverse_transform, dsize: (output_size[1], output_size[0])) # OpenCV uses (width, height)

    else:  # Projective transformation (Homography, 3x3 matrix)
        # Invert the 3x3 homography matrix
        inverse_transform = np.linalg.inv(original_transform).astype(np.float32)
        # Use cv2.warpPerspective for projective transformations
        transformed_image = cv2.warpPerspective(target_img, inverse_transform, dsize: (output_size[1], output_size[0])) # OpenCV uses (width, height)

    return transformed_image
```

סעיף ד': **מימוש 'inverse_transform_target_image'**

בפונקציה הזאת אנו מקבלים את התמונה שצריך למתוח חזרה לצורה המקורית, את מטריצת הטרנספורמציה ואת הממדים של התמונה הסופית. ניצור ממטריצת הטרנספורמציה את המטריצה ההופכית שלה ונפעיל אותה על התמונה בעזרת 'cv2.warpAffine' לתמונה אפינית ו-'cv2.warpPerspective' לתמונה הומולוגית.

סעיף ה': **מימוש 'stitch'**

בפונקציה הזאת אנו מקבלים את התמונה שצריך להוסיף (img2) לתמונה הראשונית (img1). על מנת להימנע מקווים שחורים בין התמונות ניצור מסכה כך שכל פיקסל שערכו 0 או אחד השכנים שלו הוא אפס לא ניקח אותו להדבקה על התמונה הראשונית. את השאר נדביק על התמונה הראשונית.

```
def stitch(img1, img2):  1 usage
    # Create a binary mask where img2 has non-zero pixels
    mask = (img2 > 0).any(axis=-1)  # Shape: (H, W)

    # Pad the mask to handle boundary conditions
    padded_mask = np.pad(mask, ((1, 1), (1, 1)), mode='constant', constant_values=0)

    # Check the neighbors in the 3x3 region
    neighbor_mask = (
            padded_mask[:-2, :-2] & padded_mask[:-2, 1:-1] & padded_mask[:-2, 2:] &  # Top row
            padded_mask[1:-1, :-2] & padded_mask[1:-1, 1:-1] & padded_mask[1:-1, 2:] &  # Middle row
            padded_mask[2:, :-2] & padded_mask[2:, 1:-1] & padded_mask[2:, 2:]  # Bottom row
    )

    # Combine the neighbor mask with the original mask
    valid_mask = neighbor_mask & mask

    # Use the valid mask to prioritize img2
    stitched_image = np.where(valid_mask[..., None], img2, img1)

    return stitched_image
```

סעיף ו': **הדפסת התמונה הסופית**

לאחר שיחזור כל החתיכות והדבקתן על התמונה הראשונה נציג את התמונה החדשה של כל התמונות ביחד.