BIS_1BM22CS275

Application:

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from google.colab import files

# Step 1: Upload images
uploaded = files.upload()

# Check if both images are uploaded
if len(uploaded) < 2:
    print("Please upload exactly two images.")
    exit()

# Load the images
fixed_image = cv2.imread(list(uploaded.keys())[0], cv2.IMREAD_GRAYSCALE)
# Fixed image
moving_image = cv2.imread(list(uploaded.keys())[1], cv2.IMREAD_GRAYSCALE)
# Moving image

# Check if images are loaded correctly
if fixed_image is None or moving_image is None:
    print("Error loading images. Please make sure they are valid.")
    exit()

# Resize images to the same dimensions
fixed_image = cv2.resize(fixed_image, (500, 500))
moving_image = cv2.resize(moving_image, (500, 500))

# Normalize pixel values
fixed_image = fixed_image / 255.0
moving_image = moving_image / 255.0

print(f"Fixed image shape: {fixed_image.shape}")
print(f"Moving image shape: {moving_image.shape}")

# Step 2: Define the Objective Function (Mean Squared Error)
```

```
def objective_function(params, fixed_image, moving_image):
    tx, ty, theta = params
    M = cv2.getRotationMatrix2D((moving_image.shape[1] / 2,
moving_image.shape[0] / 2), theta, 1)
    M[0, 2] += tx
    M[1, 2] += ty
    transformed_image = cv2.warpAffine(moving_image, M,
(moving_image.shape[1], moving_image.shape[0]))

    # Compute MSE only on valid regions
    valid_mask = (transformed_image > 0) & (fixed_image > 0)
    mse = np.mean((transformed_image[valid_mask] -
fixed_image[valid_mask]) ** 2)
    return mse

# Step 3: Grey Wolf Optimizer (GWO) for Image Registration
def grey_wolf_optimizer(fixed_image, moving_image, num_wolves=5,
num_iterations=30, lb=[-10, -10, -45], ub=[10, 10, 45]):
    wolves = np.random.uniform(lb, ub, (num_wolves, 3))  # Initialize
population
    alpha_pos, beta_pos, delta_pos = np.zeros(3), np.zeros(3), np.zeros(3)
    alpha_score, beta_score, delta_score = float('inf'), float('inf'),
float('inf')
    alpha_score_history = []

    # Evaluate fitness and assign alpha, beta, delta wolves
    def evaluate_fitness():
        nonlocal alpha_pos, beta_pos, delta_pos, alpha_score, beta_score,
delta_score
        for wolf in wolves:
            fitness = objective_function(wolf, fixed_image, moving_image)
            if fitness < alpha_score:
                delta_score, delta_pos = beta_score, beta_pos.copy()
                beta_score, beta_pos = alpha_score, alpha_pos.copy()
                alpha_score, alpha_pos = fitness, wolf.copy()
            elif fitness < beta_score:
                delta_score, delta_pos = beta_score, beta_pos.copy()
                beta_score, beta_pos = fitness, wolf.copy()
            elif fitness < delta_score:
                delta_score, delta_pos = fitness, wolf.copy()
```

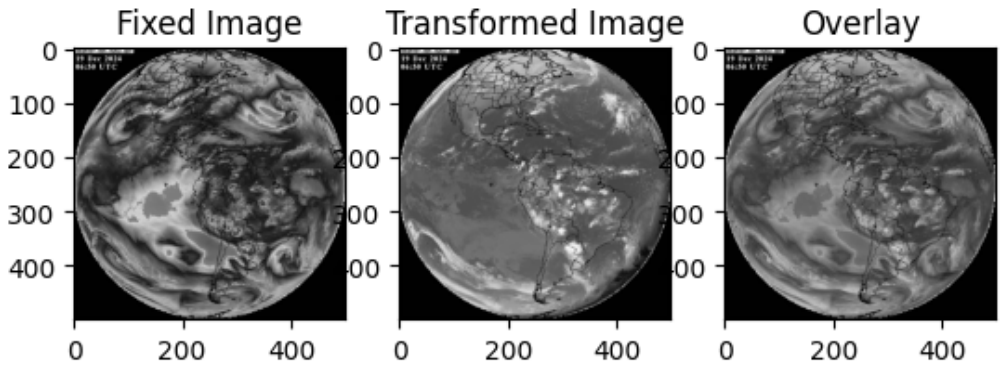
```
    # Update positions of the wolves
    def update_positions(iteration):
        a = 2 - iteration * (2 / num_iterations)  # Linearly decreasing
coefficient
        for i in range(num_wolves):
            for j in range(3):  # Update each parameter (tx, ty, theta)
                r1, r2 = np.random.random(), np.random.random()
                A1, C1 = 2 * a * r1 - a, 2 * r2
                D_alpha = abs(C1 * alpha_pos[j] - wolves[i, j])
                X1 = alpha_pos[j] - A1 * D_alpha

                r1, r2 = np.random.random(), np.random.random()
                A2, C2 = 2 * a * r1 - a, 2 * r2
                D_beta = abs(C2 * beta_pos[j] - wolves[i, j])
                X2 = beta_pos[j] - A2 * D_beta

                r1, r2 = np.random.random(), np.random.random()
                A3, C3 = 2 * a * r1 - a, 2 * r2
                D_delta = abs(C3 * delta_pos[j] - wolves[i, j])
                X3 = delta_pos[j] - A3 * D_delta

                wolves[i, j] = (X1 + X2 + X3) / 3  # Update position
                wolves[i, j] = np.clip(wolves[i, j], lb[j], ub[j])  #
Apply bounds

    # Main optimization loop
    for iteration in range(num_iterations):
        evaluate_fitness()
        update_positions(iteration)
        alpha_score_history.append(alpha_score)
        print(f"Iteration {iteration + 1}/{num_iterations}, Alpha Score:
{alpha_score}")

    # Best solution
    print("Best Transformation Parameters:", alpha_pos)
    print("Best Fitness (MSE):", alpha_score)

    # Apply the best transformation
    tx, ty, theta = alpha_pos
```

```
    M = cv2.getRotationMatrix2D((moving_image.shape[1] / 2,
moving_image.shape[0] / 2), theta, 1)
    M[0, 2] += tx
    M[1, 2] += ty
    transformed_image = cv2.warpAffine(moving_image, M,
(moving_image.shape[1], moving_image.shape[0]))

    # Visualize the results
    plt.subplot(1, 3, 1)
    plt.imshow(fixed_image, cmap='gray')
    plt.title('Fixed Image')

    plt.subplot(1, 3, 2)
    plt.imshow(transformed_image, cmap='gray')
    plt.title('Transformed Image')

    plt.subplot(1, 3, 3)
    overlay = cv2.addWeighted(fixed_image, 0.5, transformed_image, 0.5, 0)
    plt.imshow(overlay, cmap='gray')
    plt.title('Overlay')

    plt.show()

    return alpha_pos

# Perform image registration using GWO
best_params = grey_wolf_optimizer(fixed_image, moving_image,
num_wolves=10, num_iterations=50, lb=[-20, -20, -30], ub=[20, 20, 30])
```

Output:

Iteration 7/50, Alpha Score: 0.049041418317253026
Iteration 8/50, Alpha Score: 0.04885454584214083
Iteration 9/50, Alpha Score: 0.04816693466280561
Iteration 10/50, Alpha Score: 0.04386720904389379
Iteration 11/50, Alpha Score: 0.04386720904389379
Iteration 12/50, Alpha Score: 0.03904569467765534
Iteration 13/50, Alpha Score: 0.03554081905798889
Iteration 14/50, Alpha Score: 0.03396070910456307
Iteration 15/50, Alpha Score: 0.03396070910456307
Iteration 16/50, Alpha Score: 0.03331370681419242
Iteration 17/50, Alpha Score: 0.03331324148997143

Iteration 18/50, Alpha Score: 0.03329163126304798
Iteration 19/50, Alpha Score: 0.03327527017262847
Iteration 20/50, Alpha Score: 0.033269778358965434
Iteration 21/50, Alpha Score: 0.03326623989575386
Iteration 22/50, Alpha Score: 0.03326623989575386
Iteration 23/50, Alpha Score: 0.03326623989575386
Iteration 24/50, Alpha Score: 0.03326623989575386
Iteration 25/50, Alpha Score: 0.03326623989575386
Iteration 26/50, Alpha Score: 0.03326623989575386
Iteration 27/50, Alpha Score: 0.03326623989575386
Iteration 28/50, Alpha Score: 0.03326623989575386
Iteration 29/50, Alpha Score: 0.03326623989575386
Iteration 30/50, Alpha Score: 0.03326623989575386
Iteration 31/50, Alpha Score: 0.033264352574148774
Iteration 32/50, Alpha Score: 0.033264352574148774
Iteration 33/50, Alpha Score: 0.033264352574148774
Iteration 34/50, Alpha Score: 0.033264352574148774
Iteration 35/50, Alpha Score: 0.033264352574148774
Iteration 36/50, Alpha Score: 0.033264352574148774
Iteration 37/50, Alpha Score: 0.033264352574148774
Iteration 38/50, Alpha Score: 0.03326422322775631
Iteration 39/50, Alpha Score: 0.03326418735925407
Iteration 40/50, Alpha Score: 0.03326359535053911
Iteration 41/50, Alpha Score: 0.03326258679338607
Iteration 42/50, Alpha Score: 0.03326258679338607
Iteration 43/50, Alpha Score: 0.03326258679338607
Iteration 44/50, Alpha Score: 0.03326258679338607
Iteration 45/50, Alpha Score: 0.03326258679338607
Iteration 46/50, Alpha Score: 0.033262363749936914
Iteration 47/50, Alpha Score: 0.03326190128607506
Iteration 48/50, Alpha Score: 0.033261825347851694
Iteration 49/50, Alpha Score: 0.03326149742568916
Iteration 50/50, Alpha Score: 0.03326102993389504
Best Transformation Parameters: [-0.11466809 0.11678014 -0.01009706]
Best Fitness (MSE): 0.03326102993389504


Fixed Image
Transformed Image
Overlay
0
100
200
300
400
0
200
400