```python
import numpy as np

def grey_wolf_optimizer(obj_function, dim, n_wolves, lb, ub, max_iter):
    """
    Grey Wolf Optimizer (GWO) implementation.

    Parameters:
    obj_function: callable
        The objective function to minimize.
    dim: int
        The number of dimensions of the search space.
    n_wolves: int
        The number of wolves in the pack.
    lb: float or array_like
        The lower bound of the search space.
    ub: float or array_like
        The upper bound of the search space.
    max_iter: int
        The maximum number of iterations.

    Returns:
    dict
        A dictionary containing the best solution and its fitness value.
    """
    # Initialize positions of wolves
    positions = np.random.uniform(low=lb, high=ub, size=(n_wolves, dim))

    # Initialize Alpha, Beta, and Delta positions and scores
    Alpha_pos = np.zeros(dim)
    Alpha_score = float("inf")

    Beta_pos = np.zeros(dim)
    Beta_score = float("inf")

    Delta_pos = np.zeros(dim)
    Delta_score = float("inf")

    # Iterative optimization process
    for t in range(max_iter):
```

```python
    # Iterative optimization process
    for t in range(max_iter):
        for i in range(n_wolves):
            # Calculate fitness of each wolf
            fitness = obj_function(positions[i, :])

            # Update Alpha, Beta, and Delta
            if fitness < Alpha_score:
                Alpha_score = fitness
                Alpha_pos = positions[i, :].copy()

            elif fitness < Be (variable) fitness: Any
                Beta_score = fitness
                Beta_pos = positions[i, :].copy()

            elif fitness < Delta_score:
                Delta_score = fitness
                Delta_pos = positions[i, :].copy()

        # Update position of each wolf
        a = 2 - t * (2 / max_iter)  # Decreasing factor

        for i in range(n_wolves):
            for j in range(dim):
                r1, r2 = np.random.rand(), np.random.rand()
                A1 = 2 * a * r1 - a
                C1 = 2 * r2

                D_alpha = abs(C1 * Alpha_pos[j] - positions[i, j])
                X1 = Alpha_pos[j] - A1 * D_alpha

                r1, r2 = np.random.rand(), np.random.rand()
                A2 = 2 * a * r1 - a
                C2 = 2 * r2

                D_beta = abs(C2 * Beta_pos[j] - positions[i, j])
                X2 = Beta_pos[j] - A2 * D_beta

                r1, r2 = np.random.rand(), np.random.rand()
```

```python
                D_beta = abs(C2 * Beta_pos[j] - positions[i, j])
                X2 = Beta_pos[j] - A2 * D_beta

                r1, r2 = np.random.rand(), np.random.rand()
                A3 = 2 * a * r1 - a
                C3 = 2 * r2

                D_delta = abs(C3 * Delta_pos[j] - positions[i, j])
                X3 = Delta_pos[j] - A3 * D_delta

                # Update position
                positions[i, j] = (X1 + X2 + X3) / 3

            # Apply boundary constraints
            positions[i, :] = np.clip(positions[i, :], lb, ub)

    return {"best_position": Alpha_pos, "best_score": Alpha_score}

# Example usage
def sphere_function(x):
    return sum(x**2)

# Define parameters
dim = 5
n_wolves = 20
lb = -10
ub = 10
max_iter = 100

# Run the GWO algorithm
result = grey_wolf_optimizer(sphere_function, dim, n_wolves, lb, ub, max_iter)
print("Best Position:", result["best_position"])
print("Best Score:", result["best_score"])
```

```
Best Position: [ 1.11452439e-08  1.00141573e-08 -1.17124797e-08  1.10058285e-08
 -9.72603074e-09]
Best Score: 5.774059247534017e-16
```

```python
import numpy as np
import cv2

def mse(image1, image2):
    """
    Compute the Mean Squared Error (MSE) between two images.
    """
    return np.mean((image1 - image2) ** 2)

def transform_image(image, s, theta, tx, ty):
    """
    Apply scaling, rotation, and translation to an image.
    """
    # Get image center
    (h, w) = image.shape[:2]
    center = (w // 2, h // 2)

    # Create transformation matrix
    scale_matrix = cv2.getRotationMatrix2D(center, theta, s)
    scale_matrix[0, 2] += tx
    scale_matrix[1, 2] += ty

    # Apply transformation
    transformed = cv2.warpAffine(image, scale_matrix, (w, h))
    return transformed

def alignment_objective(params, ref_image, moving_image):
    """
    Objective function to optimize using GWO.
    params: [s, theta, tx, ty]
    """
    s, theta, tx, ty = params
    transformed_image = transform_image(moving_image, s, theta, tx, ty)
    return mse(ref_image, transformed_image)

def optimize_image_alignment(ref_image, moving_image, lb, ub, n_wolves=10, max_iter=100):
    """
    Optimize image alignment parameters using Grey Wolf Optimizer.
    """
    dim = 4  # [s, theta, tx, ty]
```

```python
    def obj_function(params):
        return alignment_objective(params, ref_image, moving_image)

    return grey_wolf_optimizer(obj_function, dim, n_wolves, lb, ub, max_iter)

def resize_image(image, target_shape):
    """
    Resize an image to the target shape.
    """
    return cv2.resize(image, (target_shape[1], target_shape[0]))

if __name__ == "__main__":
    # Load images (convert to grayscale for simplicity)
    ref_image = cv2.imread("/content/images.jpg", cv2.IMREAD_GRAYSCALE)
    moving_image = cv2.imread("/content/rename.jpg", cv2.IMREAD_GRAYSCALE)

    # Resize the moving image to match the reference image
    moving_image = resize_image(moving_image, ref_image.shape)

    # Define bounds for [s, theta, tx, ty]
    lb = [0.5, -180, -50, -50]  # Scale, Rotation (deg), Translation X, Translation Y
    ub = [2.0, 180, 50, 50]

    # Run optimization
    result = optimize_image_alignment(ref_image, moving_image, lb, ub, n_wolves=20, max_iter=100)
    best_params = result["best_position"]
    print("Best Parameters (s, theta, tx, ty):", best_params)

    # Apply the best transformation
    aligned_image = transform_image(moving_image, *best_params)

    # Display results
    import matplotlib.pyplot as plt
    plt.figure(figsize=(12, 6))
    plt.subplot(131)
    plt.imshow(ref_image, cmap='gray')
    plt.title('Reference Image')
    plt.subplot(132)
    plt.imshow(moving_image, cmap='gray')
    plt.title('Moving Image')
```

```python
# Display results
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 6))
plt.subplot(131)
plt.imshow(ref_image, cmap='gray')
plt.title('Reference Image')
plt.subplot(132)
plt.imshow(moving_image, cmap='gray')
plt.title('Moving Image')
plt.subplot(133)
plt.imshow(aligned_image, cmap='gray')
plt.title('Aligned Image')
```

```
Best Parameters (s, theta, tx, ty): [ 0.87119057 -6.06283112  0.16976659 -0.32639546]
```