**My Algorithm:**

My code has several limitations and errors, so you will need to write your own code. One key issue is that I used fixed parameters, which should instead be dynamic and adaptive. My current implementation only works with my specific stereo image and fails with other images. Please ensure your code is more general and flexible to work with a variety of inputs.

import csv  
import cv2  
import numpy as np  
from skimage import measure  
  
  
def mse(blockA, blockB):  
 if blockA.size == 0 or blockB.size == 0 or blockA.shape != blockB.shape:  
 return float('inf') # Assign a high error if blocks are invalid  
 err = np.sum((blockA - blockB) \*\* 2)  
 err /= float(blockA.shape[0] \* blockA.shape[1])  
 return err  
  
def calculate\_stddev(block):  
 if block.size == 0:  
 return float('inf') # Return a high stddev for invalid blocks  
 return np.std(block)  
  
  
def read\_parameters(filename='parameters.txt'):  
 params = {}  
 with open(filename, 'r') as file:  
 for line in file:  
 name, value = line.strip().split('=')  
 if name in ['left\_image', 'right\_image']:  
 params[name] = value  
 else:  
 params[name] = float(value) if '.' in value else int(value)  
 return params  
  
def apply\_colormap(map\_data):  
 normalized\_map = cv2.normalize(map\_data, None, 0, 255, cv2.NORM\_MINMAX)  
 normalized\_map = np.uint8(normalized\_map)  
 color\_map = cv2.applyColorMap(normalized\_map, cv2.COLORMAP\_JET)  
 return normalized\_map, color\_map  
  
def save\_maps(grayscale\_map, color\_map, grayscale\_filename, color\_filename):  
 cv2.imwrite(grayscale\_filename, grayscale\_map)  
 cv2.imwrite(color\_filename, color\_map)  
  
def write\_matching\_info\_to\_csv(csv\_filename, x\_left, y\_left, x\_right, y\_right, disparity, mse, direction=None, distance=None, block\_size=None):  
 with open(csv\_filename, mode='a', newline='') as file:  
 writer = csv.writer(file)  
 writer.writerow([x\_left, y\_left, x\_right, y\_right, disparity, mse, direction, distance, block\_size])  
  
def hierarchical\_block\_matching(left\_image, right\_image, params):  
 height, width = left\_image.shape[:2]  
 disparity\_map = np.zeros\_like(left\_image, dtype=np.float32)  
  
 initial\_block\_size = params['initial\_block\_size']  
 min\_block\_size = params['min\_block\_size']  
 max\_disparity = params['max\_disparity']  
 min\_disparity = params['min\_disparity']  
 max\_right\_search = params['max\_right\_search']  
 max\_other\_directions\_search = params['max\_other\_directions\_search']  
 max\_mse = params['max\_mse']  
  
 directions = [  
 ((0, 1), 'Right'), # Right  
 ((1, 1), 'Down-Right'), # Down-Right  
 ((1, 0), 'Down'), # Down  
 ((1, -1), 'Down-Left'), # Down-Left  
 ((0, -1), 'Left'), # Left  
 ((-1, -1), 'Up-Left'), # Up-Left  
 ((-1, 0), 'Up'), # Up  
 ((-1, 1), 'Up-Right') # Up-Right  
 ]  
  
 csv\_filename = 'matching\_blocks.csv'  
 with open(csv\_filename, mode='w', newline='') as file:  
 writer = csv.writer(file)  
 writer.writerow(['x\_left', 'y\_left', 'x\_right', 'y\_right', 'disparity', 'mse', 'direction', 'distance', 'block\_size'])  
  
 def pixel\_by\_pixel\_search(x, y, block\_size):  
 MSEMin = 1000000 # initially high  
 best\_mse = float('inf') # initially high  
 best\_match = None  
 best\_disparity = 0  
  
 for depth in range(1, max(max\_right\_search, max\_disparity, max\_other\_directions\_search) + 1):  
 for (dy, dx), direction\_name in directions:  
 if direction\_name == 'Right' and depth > max\_right\_search:  
 continue  
 elif direction\_name == 'Left' and depth > max\_disparity:  
 continue  
 elif direction\_name not in ['Right', 'Left'] and depth > max\_other\_directions\_search:  
 continue  
  
 nx = x + dx \* depth  
 ny = y + dy \* depth  
  
 if nx < 0 or nx + block\_size > width or ny < 0 or ny + block\_size > height:  
 continue  
  
 ref\_block = left\_image[y:y + block\_size, x:x + block\_size]  
 search\_block = right\_image[ny:ny + block\_size, nx:nx + block\_size]  
  
 if search\_block.shape != ref\_block.shape:  
 continue  
  
 error = mse(ref\_block, search\_block)  
 stddev = calculate\_stddev(search\_block)  
  
 if stddev < params['std\_dev\_threshold']:  
 max\_mse = params['max\_mse'] / 2 # For a lower texture, lower the max allowed mse for good matches  
 else:  
 max\_mse = params['max\_mse']  
  
 MSEMin = min(MSEMin, error) # update the minimum MSE value  
  
 if MSEMin < best\_mse:  
 best\_mse = MSEMin  
 best\_disparity = depth  
 best\_match = search\_block  
  
 if best\_match is not None and MSEMin <= max\_mse:  
 disparity\_map[y:y + block\_size, x:x + block\_size] = best\_disparity  
 grayscale\_map, color\_map = apply\_colormap(disparity\_map)  
 save\_maps(grayscale\_map, color\_map, 'grayscale\_disparity\_map.png', 'color\_disparity\_map.png')  
 write\_matching\_info\_to\_csv(csv\_filename, x, y, x - best\_disparity, y, best\_disparity, best\_mse,  
 direction\_name, depth, block\_size=block\_size)  
 return True  
  
 disparity\_map[y:y + block\_size, x:x + block\_size] = 0 # set block as black if MSEMin is greater than max\_mse  
 return False  
  
 def split\_and\_match\_block(x, y, block\_size):  
 if block\_size < min\_block\_size:  
 return False  
  
 ref\_block = left\_image[y:y + block\_size, x:x + block\_size]  
 best\_match = None  
 best\_mse = float('inf')  
 best\_disparity = 0  
  
 for disparity in range(min\_disparity, max\_disparity):  
 if x - disparity < 0:  
 continue  
 if x - disparity + block\_size > width:  
 continue  
  
 search\_block = right\_image[y:y + block\_size, x - disparity:x - disparity + block\_size]  
 if search\_block.shape != ref\_block.shape:  
 continue  
 error = mse(ref\_block, search\_block)  
 stddev = calculate\_stddev(ref\_block)  
  
 if stddev < params['std\_dev\_threshold']:  
 max\_mse = params['max\_mse'] / 2  
 else:  
 max\_mse = params['max\_mse']  
  
 if error < best\_mse and error <= max\_mse:  
 best\_mse = error  
 best\_disparity = disparity  
 best\_match = search\_block  
  
 if best\_match is not None:  
 disparity\_map[y:y + block\_size, x:x + block\_size] = best\_disparity  
 grayscale\_map, color\_map = apply\_colormap(disparity\_map)  
 save\_maps(grayscale\_map, color\_map, 'grayscale\_disparity\_map.png', 'color\_disparity\_map.png')  
 write\_matching\_info\_to\_csv(csv\_filename, x, y, x - best\_disparity, y, best\_disparity, best\_mse, block\_size=block\_size)  
 return True  
  
 if pixel\_by\_pixel\_search(x, y, block\_size):  
 return True  
  
 half\_block\_size = block\_size // 2  
 if half\_block\_size >= min\_block\_size:  
 split\_and\_match\_block(x, y, half\_block\_size)  
 split\_and\_match\_block(x + half\_block\_size, y, half\_block\_size)  
 split\_and\_match\_block(x, y + half\_block\_size, half\_block\_size)  
 split\_and\_match\_block(x + half\_block\_size, y + half\_block\_size, half\_block\_size)  
  
 return False  
  
 for y in range(0, height, initial\_block\_size):  
 for x in range(0, width, initial\_block\_size):  
 split\_and\_match\_block(x, y, initial\_block\_size)  
  
 return disparity\_map  
  
def save\_low\_mse\_low\_texture\_blocks\_to\_csv(left\_image, right\_image, params, csv\_filename='low\_mse\_low\_texture\_blocks.csv'):  
 height, width = left\_image.shape[:2]  
 with open(csv\_filename, mode='w', newline='') as file:  
 writer = csv.writer(file)  
 writer.writerow(['x\_left', 'y\_left', 'x\_right', 'y\_right', 'mse', 'stddev'])  
  
 block\_size = params['initial\_block\_size']  
 for y in range(0, height, block\_size):  
 for x in range(0, width, block\_size):  
 ref\_block = left\_image[y:y + block\_size, x:x + block\_size]  
 best\_match = None  
 best\_mse = float('inf')  
 best\_disparity = 0  
  
 for disparity in range(params['min\_disparity'], params['max\_disparity']):  
 if x - disparity < 0:  
 continue  
 if x - disparity + block\_size > width:  
 continue  
  
 search\_block = right\_image[y:y + block\_size, x - disparity:x - disparity + block\_size]  
 if search\_block.shape != ref\_block.shape:  
 continue  
 error = mse(ref\_block, search\_block)  
 stddev = calculate\_stddev(ref\_block)  
  
 if error < best\_mse and stddev < params['std\_dev\_threshold']:  
 best\_mse = error  
 best\_disparity = disparity  
 best\_match = search\_block  
  
 if best\_match is not None and best\_mse <= params['max\_mse'] and calculate\_stddev(best\_match) < params['std\_dev\_threshold']:  
 writer.writerow([int(x), int(y), int(x - best\_disparity), int(y), best\_mse, calculate\_stddev(best\_match)])  
  
def create\_low\_mse\_low\_texture\_maps(left\_image, right\_image, csv\_filename='low\_mse\_low\_texture\_blocks.csv'):  
 height, width = left\_image.shape[:2]  
 left\_map = np.ones((height, width), dtype=np.uint8) \* 255 # Initialize as white image  
 right\_map = np.ones((height, width), dtype=np.uint8) \* 255 # Initialize as white image  
  
 with open(csv\_filename, mode='r') as file:  
 reader = csv.reader(file)  
 next(reader) # Skip header  
  
 for row in reader:  
 x\_left, y\_left, x\_right, y\_right, \_, \_ = map(float, row[:6])  
 x\_left, y\_left, x\_right, y\_right = int(x\_left), int(y\_left), int(x\_right), int(y\_right)  
 block\_size = params['initial\_block\_size']  
  
 # Mark the left block in black  
 left\_map[y\_left:y\_left + block\_size, x\_left:x\_left + block\_size] = 0  
  
 # Mark the right block in black  
 right\_map[y\_right:y\_right + block\_size, x\_right:x\_right + block\_size] = 0  
  
 # Save the resulting maps  
 cv2.imwrite('left\_low\_mse\_low\_texture\_map.png', left\_map)  
 cv2.imwrite('right\_low\_mse\_low\_texture\_map.png', right\_map)  
  
params = read\_parameters('parameters.txt')  
left\_image = cv2.imread(params['left\_image'])  
right\_image = cv2.imread(params['right\_image'])  
  
# Convert images to grayscale  
left\_image = cv2.cvtColor(left\_image, cv2.COLOR\_BGR2GRAY)  
right\_image = cv2.cvtColor(right\_image, cv2.COLOR\_BGR2GRAY)  
  
disparity\_map = hierarchical\_block\_matching(left\_image, right\_image, params)  
save\_low\_mse\_low\_texture\_blocks\_to\_csv(left\_image, right\_image, params)  
create\_low\_mse\_low\_texture\_maps(left\_image, right\_image)  
  
def write\_pixel\_data\_to\_csv(csv\_filename, x\_left, y\_left, x\_right, y\_right, left\_pixel, right\_pixel, mse, stddev):  
 with open(csv\_filename, mode='a', newline='') as file:  
 writer = csv.writer(file)  
 writer.writerow([x\_left, y\_left, x\_right, y\_right, left\_pixel, right\_pixel, mse, stddev])  
  
  
def save\_low\_mse\_low\_texture\_pixels\_to\_csv(left\_image, right\_image, params,  
 csv\_filename='low\_mse\_low\_texture\_pixels.csv'):  
 height, width = left\_image.shape[:2]  
  
 # Initialize CSV with header  
 with open(csv\_filename, mode='w', newline='') as file:  
 writer = csv.writer(file)  
 writer.writerow(  
 ['x\_left', 'y\_left', 'x\_right', 'y\_right', 'left\_pixel\_value', 'right\_pixel\_value', 'mse', 'stddev'])  
  
 block\_size = params['initial\_block\_size']  
  
 for y in range(0, height, block\_size):  
 for x in range(0, width, block\_size):  
 ref\_block = left\_image[y:y + block\_size, x:x + block\_size]  
 best\_match = None  
 best\_mse = float('inf')  
 best\_disparity = 0  
  
 # Iterate through disparity to find the best match  
 for disparity in range(params['min\_disparity'], params['max\_disparity']):  
 if x - disparity < 0:  
 continue  
 if x - disparity + block\_size > width:  
 continue  
  
 search\_block = right\_image[y:y + block\_size, x - disparity:x - disparity + block\_size]  
 if search\_block.shape != ref\_block.shape:  
 continue  
  
 # Calculate MSE and standard deviation  
 error = mse(ref\_block, search\_block)  
 stddev = calculate\_stddev(ref\_block)  
  
 # Check if this block has low MSE and low texture  
 if error < best\_mse and stddev < params['std\_dev\_threshold']:  
 best\_mse = error  
 best\_disparity = disparity  
 best\_match = search\_block  
  
 # If a match with low MSE and low texture is found, save each pixel data  
 if best\_match is not None and best\_mse <= params['max\_mse'] and stddev < params['std\_dev\_threshold']:  
 for i in range(block\_size):  
 for j in range(block\_size):  
 # Get pixel coordinates and values  
 x\_left\_pixel = x + j  
 y\_left\_pixel = y + i  
 x\_right\_pixel = (x - best\_disparity) + j  
 y\_right\_pixel = y + i  
 left\_pixel\_value = ref\_block[i, j]  
 right\_pixel\_value = best\_match[i, j]  
  
 # Write each pixel's information to CSV  
 write\_pixel\_data\_to\_csv(csv\_filename, x\_left\_pixel, y\_left\_pixel, x\_right\_pixel, y\_right\_pixel,  
 left\_pixel\_value, right\_pixel\_value, best\_mse, stddev)  
  
save\_low\_mse\_low\_texture\_pixels\_to\_csv(left\_image, right\_image, params)  
  
def create\_pixel\_location\_maps(csv\_filename='low\_mse\_low\_texture\_pixels.csv'):  
 height, width = left\_image.shape[:2]  
 left\_pixel\_map = np.ones((height, width), dtype=np.uint8) \* 255 # Initialize as white image  
 right\_pixel\_map = np.ones((height, width), dtype=np.uint8) \* 255 # Initialize as white image  
  
 with open(csv\_filename, mode='r') as file:  
 reader = csv.reader(file)  
 next(reader) # Skip header  
  
 for row in reader:  
 x\_left, y\_left, x\_right, y\_right, \_, \_, \_, \_ = map(float, row[:8])  
 x\_left, y\_left, x\_right, y\_right = int(x\_left), int(y\_left), int(x\_right), int(y\_right)  
  
 # Mark the left pixel in black  
 left\_pixel\_map[y\_left, x\_left] = 0  
  
 # Mark the right pixel in black  
 right\_pixel\_map[y\_right, x\_right] = 0  
  
 # Save the resulting maps  
 cv2.imwrite('left\_pixel\_location\_map.png', left\_pixel\_map)  
 cv2.imwrite('right\_pixel\_location\_map.png', right\_pixel\_map)  
  
create\_pixel\_location\_maps()  
  
  
def create\_connected\_pixel\_maps\_with\_marching\_squares(csv\_filename='low\_mse\_low\_texture\_pixels.csv'):  
 height, width = left\_image.shape[:2]  
 left\_pixel\_map = np.ones((height, width), dtype=np.uint8) \* 255 # Initialize as white image  
 right\_pixel\_map = np.ones((height, width), dtype=np.uint8) \* 255 # Initialize as white image  
  
 # Populate pixel maps from CSV file  
 with open(csv\_filename, mode='r') as file:  
 reader = csv.reader(file)  
 next(reader) # Skip header  
  
 for row in reader:  
 x\_left, y\_left, x\_right, y\_right, \_, \_, \_, \_ = map(float, row[:8])  
 x\_left, y\_left, x\_right, y\_right = int(x\_left), int(y\_left), int(x\_right), int(y\_right)  
  
 left\_pixel\_map[y\_left, x\_left] = 0 # Mark the left pixel in black  
 right\_pixel\_map[y\_right, x\_right] = 0 # Mark the right pixel in black  
  
 # Use Marching Squares to connect pixels  
 contours\_left = measure.find\_contours(left\_pixel\_map, 0.5)  
 contours\_right = measure.find\_contours(right\_pixel\_map, 0.5)  
  
 # Draw contours on maps  
 for contour in contours\_left:  
 contour = np.round(contour).astype(int)  
 for y, x in contour:  
 if 0 <= y < height and 0 <= x < width:  
 left\_pixel\_map[y, x] = 0  
  
 for contour in contours\_right:  
 contour = np.round(contour).astype(int)  
 for y, x in contour:  
 if 0 <= y < height and 0 <= x < width:  
 right\_pixel\_map[y, x] = 0  
  
 # Save the resulting maps with connected pixels  
 cv2.imwrite('left\_connected\_pixel\_map.png', left\_pixel\_map)  
 cv2.imwrite('right\_connected\_pixel\_map.png', right\_pixel\_map)  
  
create\_connected\_pixel\_maps\_with\_marching\_squares()  
  
def create\_individual\_bounding\_box\_maps(csv\_filename='low\_mse\_low\_texture\_pixels.csv'):  
 height, width = left\_image.shape[:2]  
 left\_pixel\_map = np.ones((height, width, 3), dtype=np.uint8) \* 255 # Initialize as white RGB image  
 right\_pixel\_map = np.ones((height, width, 3), dtype=np.uint8) \* 255 # Initialize as white RGB image  
  
 # Populate binary maps from CSV file to find individual sets of pixels  
 left\_binary\_map = np.zeros((height, width), dtype=np.uint8)  
 right\_binary\_map = np.zeros((height, width), dtype=np.uint8)  
  
 with open(csv\_filename, mode='r') as file:  
 reader = csv.reader(file)  
 next(reader) # Skip header  
  
 for row in reader:  
 x\_left, y\_left, x\_right, y\_right, \_, \_, \_, \_ = map(float, row[:8])  
 x\_left, y\_left, x\_right, y\_right = int(x\_left), int(y\_left), int(x\_right), int(y\_right)  
  
 left\_binary\_map[y\_left, x\_left] = 255 # Mark the left pixel  
 right\_binary\_map[y\_right, x\_right] = 255 # Mark the right pixel  
  
 # Find connected components in the left binary map  
 num\_labels\_left, labels\_left, stats\_left, \_ = cv2.connectedComponentsWithStats(left\_binary\_map, connectivity=8)  
 for i in range(1, num\_labels\_left):  
 x, y, w, h, \_ = stats\_left[i]  
 cv2.rectangle(left\_pixel\_map, (x, y), (x + w, y + h), (0, 255, 0), 2) # Draw green bounding box  
  
 # Find connected components in the right binary map  
 num\_labels\_right, labels\_right, stats\_right, \_ = cv2.connectedComponentsWithStats(right\_binary\_map, connectivity=8)  
 for i in range(1, num\_labels\_right):  
 x, y, w, h, \_ = stats\_right[i]  
 cv2.rectangle(right\_pixel\_map, (x, y), (x + w, y + h), (0, 255, 0), 2) # Draw green bounding box  
  
 # Save the resulting maps with bounding boxes  
 cv2.imwrite('left\_individual\_bounding\_box\_map.png', left\_pixel\_map)  
 cv2.imwrite('right\_individual\_bounding\_box\_map.png', right\_pixel\_map)  
  
  
create\_individual\_bounding\_box\_maps()  
  
def compare\_corresponding\_bounding\_boxes(csv\_filename='bounding\_box\_comparison.csv'):  
 height, width = left\_image.shape[:2]  
 left\_binary\_map = np.zeros((height, width), dtype=np.uint8)  
 right\_binary\_map = np.zeros((height, width), dtype=np.uint8)  
  
 # Populate binary maps from CSV file to find individual sets of pixels  
 with open('low\_mse\_low\_texture\_pixels.csv', mode='r') as file:  
 reader = csv.reader(file)  
 next(reader) # Skip header  
  
 for row in reader:  
 x\_left, y\_left, x\_right, y\_right, \_, \_, \_, \_ = map(float, row[:8])  
 x\_left, y\_left, x\_right, y\_right = int(x\_left), int(y\_left), int(x\_right), int(y\_right)  
  
 left\_binary\_map[y\_left, x\_left] = 255 # Mark the left pixel  
 right\_binary\_map[y\_right, x\_right] = 255 # Mark the right pixel  
  
 # Find connected components and bounding boxes in left binary map  
 num\_labels\_left, labels\_left, stats\_left, \_ = cv2.connectedComponentsWithStats(left\_binary\_map, connectivity=8)  
  
 # Find connected components and bounding boxes in right binary map  
 num\_labels\_right, labels\_right, stats\_right, \_ = cv2.connectedComponentsWithStats(right\_binary\_map, connectivity=8)  
  
 # Ensure that the number of connected components is the same  
 if num\_labels\_left != num\_labels\_right:  
 print("Warning: Number of connected components in left and right images are not the same.")  
 return  
  
 # Open CSV file to save comparison information  
 with open(csv\_filename, mode='w', newline='') as file:  
 writer = csv.writer(file)  
 writer.writerow(['BoundingBox\_ID', 'Position\_Left', 'Position\_Right',  
 'Width\_Left', 'Height\_Left', 'Width\_Right', 'Height\_Right', 'Area\_Left', 'Area\_Right',  
 'Average\_Color\_Left', 'Average\_Color\_Right', 'Centroid\_Distance'])  
  
 # Iterate over each corresponding bounding box in the left and right images  
 for i in range(1, num\_labels\_left):  
 # Extract bounding box info from left image  
 x\_left, y\_left, w\_left, h\_left, area\_left = stats\_left[i]  
 avg\_color\_left = cv2.mean(left\_image[y\_left:y\_left + h\_left, x\_left:x\_left + w\_left])[:3]  
 centroid\_left = (x\_left + w\_left / 2, y\_left + h\_left / 2)  
  
 # Extract bounding box info from right image  
 x\_right, y\_right, w\_right, h\_right, area\_right = stats\_right[i]  
 avg\_color\_right = cv2.mean(right\_image[y\_right:y\_right + h\_right, x\_right:x\_right + w\_right])[:3]  
 centroid\_right = (x\_right + w\_right / 2, y\_right + h\_right / 2)  
  
 # Calculate centroid distance between left and right bounding boxes  
 centroid\_distance = np.linalg.norm(np.array(centroid\_left) - np.array(centroid\_right))  
  
 # Write comparison information to CSV  
 writer.writerow([  
 i,  
 (x\_left, y\_left), (x\_right, y\_right),  
 w\_left, h\_left, w\_right, h\_right,  
 area\_left, area\_right,  
 avg\_color\_left, avg\_color\_right,  
 centroid\_distance  
 ])  
  
compare\_corresponding\_bounding\_boxes()  
  
def update\_existing\_disparity\_maps(csv\_filename='bounding\_box\_comparison.csv'):  
 # Load the existing disparity maps  
 grayscale\_disparity\_map = cv2.imread('grayscale\_disparity\_map.png', cv2.IMREAD\_GRAYSCALE)  
 color\_disparity\_map = cv2.imread('color\_disparity\_map.png')  
  
 # Convert the grayscale disparity map to a float32 type for easier manipulation of values  
 disparity\_map = np.float32(grayscale\_disparity\_map)  
  
 # Read bounding box comparison information from CSV file  
 with open(csv\_filename, mode='r') as file:  
 reader = csv.reader(file)  
 next(reader) # Skip header  
  
 for row in reader:  
 \_, position\_left, position\_right, w\_left, h\_left, \_, \_, \_, \_, \_, \_, centroid\_distance = row  
 x\_left, y\_left = eval(position\_left) # Extract top-left position of left bounding box  
 w\_left, h\_left = int(w\_left), int(h\_left)  
 disparity\_value = float(centroid\_distance) # Disparity value from centroid distance  
  
 # Assign disparity value to all pixels within the bounding box in the left image  
 disparity\_map[y\_left:y\_left + h\_left, x\_left:x\_left + w\_left] = disparity\_value  
  
 # Normalize the updated disparity map to 0-255 for visualization  
 normalized\_disparity\_map = cv2.normalize(disparity\_map, None, 0, 255, cv2.NORM\_MINMAX)  
 updated\_grayscale\_disparity\_map = np.uint8(normalized\_disparity\_map)  
  
 # Update the color disparity map using the new grayscale disparity values  
 updated\_color\_disparity\_map = cv2.applyColorMap(updated\_grayscale\_disparity\_map, cv2.COLORMAP\_JET)  
  
 # Save the resulting updated maps  
 cv2.imwrite('updated\_grayscale\_disparity\_map.png', updated\_grayscale\_disparity\_map)  
 cv2.imwrite('updated\_color\_disparity\_map.png', updated\_color\_disparity\_map)  
  
update\_existing\_disparity\_maps()

**Parameters:** (these are the parameters I used but it should be dynamic hard coded)

left\_image=left.png

right\_image=right.png

initial\_block\_size=32

min\_block\_size=8

max\_disparity=-204

min\_disparity=-239

max\_right\_search=4

max\_other\_directions\_search=2

max\_mse=25

std\_dev\_threshold=10

**Input images:**

|  |  |
| --- | --- |
| A white cubes with colorful patterns on a green background  Description automatically generated | A white cubes on a green background  Description automatically generated |

**Output disparity map:**

**A group of black rectangular objects

Description automatically generated**

**Confident maps:**

|  |  |
| --- | --- |
| A black squares on a white background  Description automatically generated | A black square with white lines  Description automatically generated with medium confidence |

**Bounding Box Maps:**

|  |  |
| --- | --- |
| A green squares on a white background  Description automatically generated | A white rectangles with green outline  Description automatically generated |

**Updated disparity map:**

A black and white cubes

Description automatically generated