AI VIET NAM – AI COURSE 2024

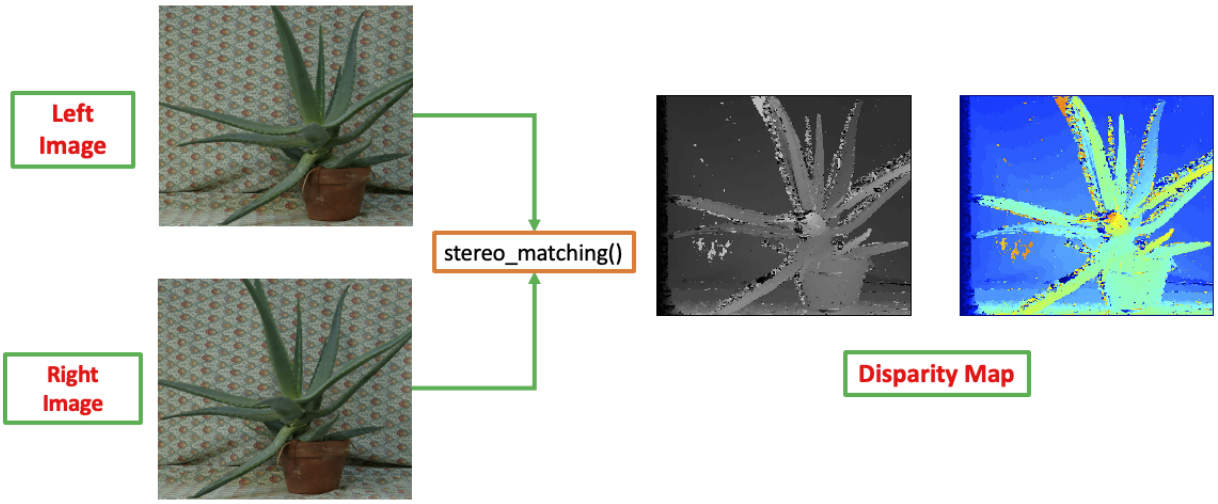
Phân tích thông tin chiều sâu của ảnh

Dinh-Thang Duong, Quang-Vinh Dinh

Ngày 2 tháng 8 năm 2024

I. Giới thiệu

Stereo Matching là một bài toán lớn trong lĩnh vực Thị giác máy tính (Computer Vision), với mục tiêu phục hồi kiến trúc 3D thực tế từ một cặp ảnh 2D, gọi là ảnh stereo. Stereo Matching thường được ứng dụng trong các ứng dụng liên quan đến Xe tự hành (Autonomous Driving), Thực tế ảo (Augmented Reality)...



Hình 1: Minh họa bài toán Depth Information Reconstruction. Kết quả bài toán là ảnh mô phỏng độ sâu (disparity map) từ một cặp ảnh.

Trong project này, các bạn sẽ làm quen với bài toán Stereo Matching thông qua việc triển khai một số thuật toán tính Disparity Map từ cặp ảnh stereo cho trước. Với Disparity Map, ta có thể biết được thông tin chiều sâu (Depth Information) thực tế. Các cặp ảnh stereo sẽ sử dụng trong project bao gồm:

- **Ảnh Tsukuba:** Sử dụng trong Problem 1. Các bạn có thể tải bộ ảnh Tsukuba tại đây.
- **Ảnh Aloe:** Sử dụng trong Problem 2, 3, 4. Các bạn có thể tải bộ ảnh Aloe tại đây.

II. Nội dung

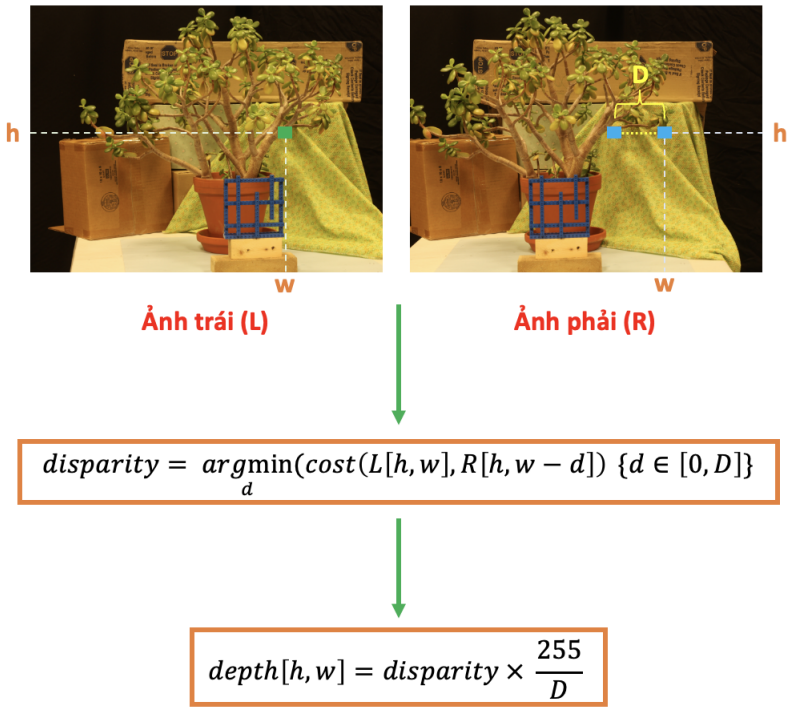
Problem 1: Xây dựng hàm tính disparity map của hai ảnh stereo đầu vào (ảnh bên trái (L) và ảnh bên phải (R)) theo phương thức **pixel-wise matching**. Các bước tính toán trong phương pháp này có thể được miêu tả qua các bước dưới đây:

1. Đọc ảnh chụp bên trái (left) và ảnh chụp bên phải (right) dưới dạng ảnh grayscale (ảnh mức xám) đồng thời ép kiểu ảnh về np.float32.
2. Khởi tạo hai biến height, width có giá trị bằng chiều cao, chiều rộng của ảnh trái.
3. Khởi tạo một ma trận không - zero matrix (depth) với kích thước bằng height, width.
4. Với mỗi pixel tại vị trí (h, w) (duyệt từ trái qua phải, trên xuống dưới) thực hiện các bước sau:
   (a) Tính cost (L1 hoặc L2) giữa các cặp pixel left[h, w] và right[h, w - d] (trong đó $d \in [0, \text{disparity\_range}]$). Nếu (w - d) < 0 thì gán giá trị cost = max_cost (max_cost = 255 nếu dùng L1 hoặc 255^2 nếu dùng L2).
   (b) Với danh sách cost tính được, chọn giá trị d ($d_{optimal}$) mà ở đó cho giá trị cost là nhỏ nhất.
   (c) Gán depth[h, w] = $d_{optimal} \times scale$. Trong đó, $scale = \frac{255}{disparity\_range}$ (Ở Problem 01, các bạn gán mặc định giá trị $scale = 16$).

Dựa theo mô tả các bước triển khai thuật toán, các bạn có thể cài đặt code như sau:

```python
import cv2
import numpy as np

def distance(x, y):
    return abs(x - y)

def pixel_wise_matching(left_img, right_img, disparity_range, save_result=True):
    # Read left, right images then convert to grayscale
    left  = cv2.imread(left_img, 0)
    right = cv2.imread(right_img, 0)

    left  = left.astype(np.float32)
    right = right.astype(np.float32)

    height, width = left.shape[:2]

    # Create blank disparity map
    depth = np.zeros((height, width), np.uint8)
    scale = 16
    max_value = 255

    for y in range(height):
        for x in range(width):
            # Find j where cost has minimum value
```

```
            disparity = 0
            cost_min  = max_value

            for j in range(disparity_range):
                cost = max_value if (x - j) < 0 else distance(int(left[y, x]),
    int(right[y, x - j]))

                if cost < cost_min:
                    cost_min  = cost
                    disparity = j

            # Let depth at (y, x) = j (disparity)
            # Multiply by a scale factor for visualization purpose
            depth[y, x] = disparity * scale

    if save_result == True:
        print('Saving result...')
        # Save results
        cv2.imwrite(f'pixel_wise_l1.png', depth)
        cv2.imwrite(f'pixel_wise_l1_color.png', cv2.applyColorMap(depth, cv2.
    COLORMAP_JET))

    print('Done.')

    return depth
```



Hình 2: Các bước tính toán giá trị disparity tại vị trí của một pixel bất kì theo phương pháp pixel-wise matching

Tận dụng phần cài đặt gợi ý trên, các bạn sẽ thực hiện thêm một số yêu cầu sau:

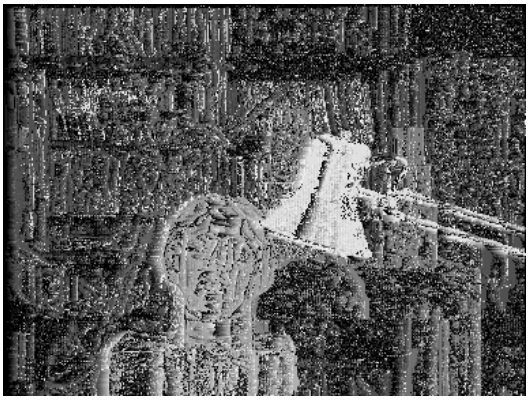
- Thiết kế hai hàm có tên gọi lần lượt là:
  - **pixel_wise_matching_l1()**
  - **pixel_wise_matching_l2()**

  Hai hàm này có chung tham số đầu vào là:
  - **left_img:** đường dẫn đến ảnh chụp bên trái.
  - **right_img:** đường dẫn đến ảnh chụp bên phải.
  - **disparity_range:** độ dài tối đa của vùng tìm kiếm giá trị disparity tại mỗi pixel. Trong Problem 01, các bạn gán mặc định giá trị này = 16.
  - **save_result:** giá trị boolean đại diện cho việc có lưu disparity map hay không? Mặc định sẽ có giá trị là True.

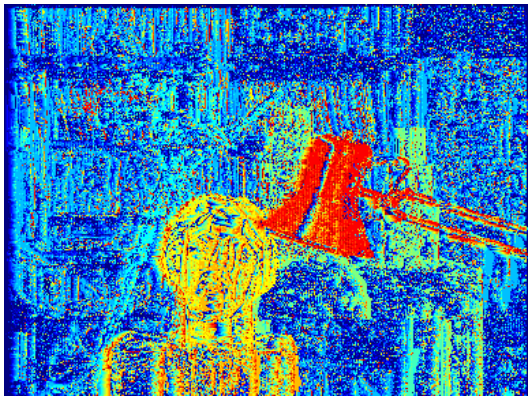
  Cuối cùng, kết quả trả về của hàm sẽ là một numpy.ndarray đại diện cho disparity map tính được.
- Đối với hàm **pixel_wise_matching_l1()**, ta dùng hàm tính cost là L1.
- Đối với hàm **pixel_wise_matching_l2()**, ta dùng hàm tính cost là L2.
- Khi lưu kết quả, ta in ra màn hình dòng chữ **"Saving result..."** và lưu ý rằng sẽ lưu cả hai phiên bản ảnh grayscale và ảnh color map của disparity map.
- Sau khi kết thúc toàn bộ quá trình tính toán, in ra dòng chữ **"Done."**.

Kết quả minh họa khi chạy hàm **pixel_wise_matching()** với cặp ảnh Tsukuba sẽ có kết quả trả về như sau (các dòng comment tượng trưng cho nội dung trả về):

```
left_img_path = 'tsukuba/left.png'
right_img_path = 'tsukuba/right.png'
disparity_range = 16

pixel_wise_result_l1 = pixel_wise_matching_l1(
    left_img_path,
    right_img_path,
    disparity_range,
    save_result=True
)

# Saving result...
# Done.

pixel_wise_result_l2 = pixel_wise_matching_l2(
    left_img_path,
    right_img_path,
    disparity_range,
    save_result=True
)

# Saving result...
# Done.
```

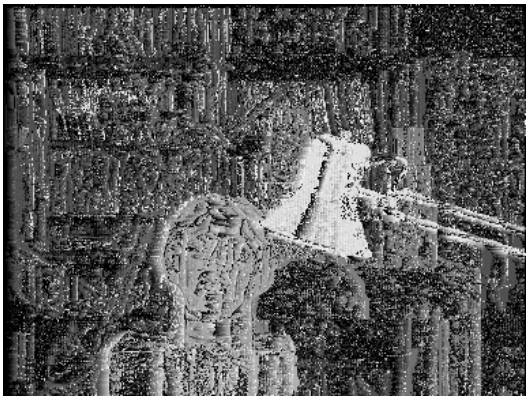


(a) Ảnh grayscale

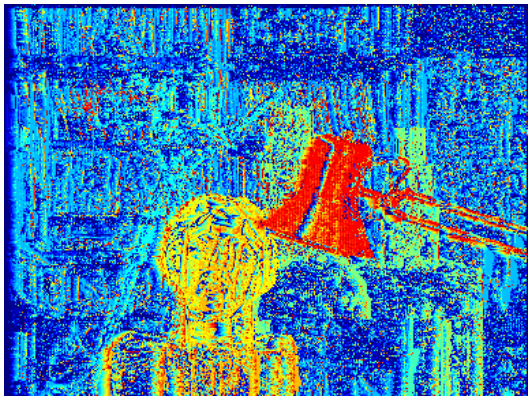


(b) Ảnh color map

Hình 3: Kết quả disparity map của phương pháp pixel-wise matching sử dụng L1



(a) Ảnh grayscale



(b) Ảnh color map

Hình 4: Kết quả disparity map của phương pháp pixel-wise matching sử dụng L2

Problem 2: Xây dựng hàm tính disparity map của hai ảnh stereo đầu vào (ảnh bên trái (L) và ảnh bên phải (R)) theo phương thức **window-based matching**. Các bước tính toán trong phương pháp này có thể được miêu tả qua các bước dưới đây:

1. Đọc ảnh chụp bên trái (left) và ảnh chụp bên phải (right) dưới dạng ảnh grayscale (ảnh mức xám) đồng thời ép kiểu ảnh về np.float32.
2. Khởi tạo hai biến height, width có giá trị bằng chiều cao, chiều rộng của ảnh trái.
3. Khởi tạo một ma trận không - zero matrix (depth) với kích thước bằng height, width.
4. Tính nửa kích thước của window tính từ tâm đến cạnh của window (có kích thước k x k) theo công thức kernel_half $= \frac{k-1}{2}$ (lấy nguyên).
5. Với mỗi pixel tại vị trí (h, w) (h $\in$ [kernel_half, height - kernel_half], w $\in$ [kernel_half, width - kernel_half]; duyệt từ trái qua phải, trên xuống dưới), thực hiện các bước sau:

(a) Tính tổng các cost (l1 hoặc l2) giữa các cặp pixel left[h + v, w + u] và right [h + v, w + u - d] (trong đó d ∈ [0, disparity_range] và u, v ∈ [-kernel_half, kernel_half]) nằm trong vùng window với tâm là vị trí của pixel đang xét. Nếu tại vị trí cho (w + u - d) < 0 thì gán giá trị cost của cặp pixel đang xét = max_cost (max_cost = 255 nếu dùng L1 hoặc 255^2 nếu dùng L2).

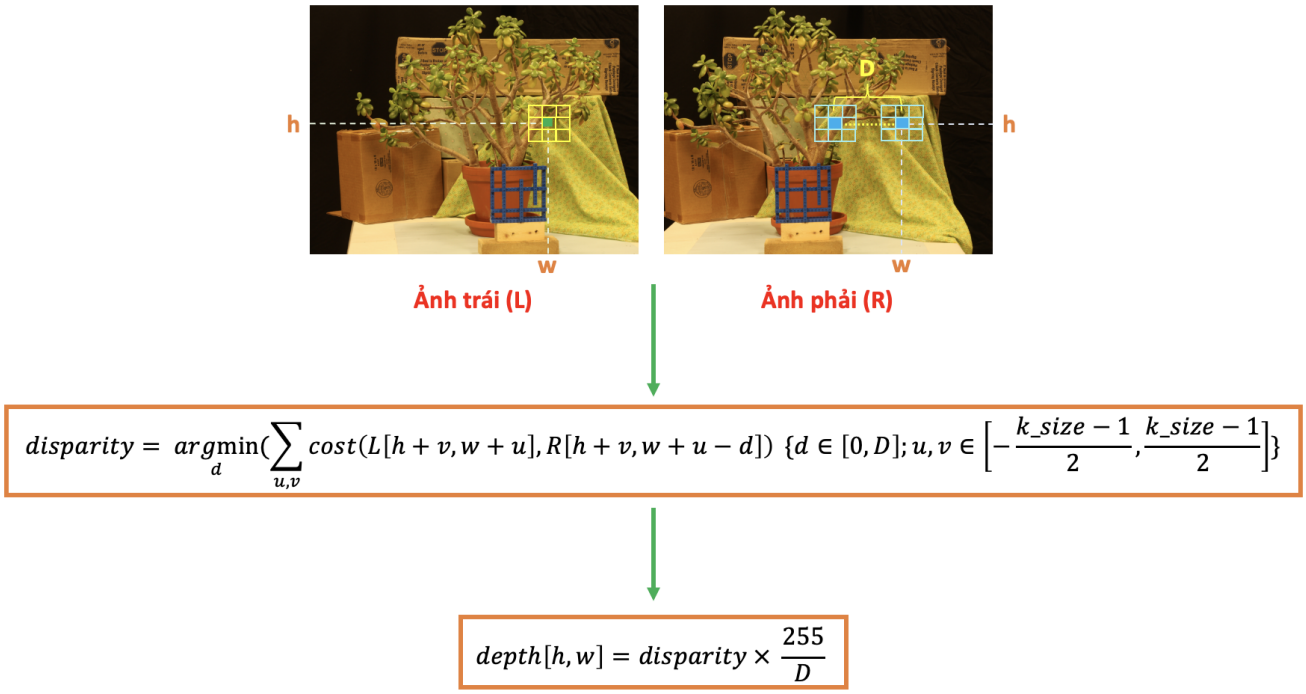
(b) Với danh sách cost tính được, chọn giá trị d ($d_{optimal}$) mà ở đó cho giá trị cost tổng là nhỏ nhất.

(c) Gán depth[h, w] = $d_{optimal} \times scale$. Trong đó, $scale = \frac{255}{disparity\_range}$ (Ở Problem 02, các bạn gán mặc định giá trị $scale = 3$).

Dựa theo mô tả các bước triển khai thuật toán, các bạn có thể cài đặt code như sau:

```
import cv2
import numpy as np

def distance(x, y):
    return abs(x - y)

def window_based_matching(left_img, right_img, disparity_range, kernel_size=5,
    save_result=True):
    # Read left, right images then convert to grayscale
    left  = cv2.imread(left_img, 0)
    right = cv2.imread(right_img, 0)

    left  = left.astype(np.float32)
    right = right.astype(np.float32)

    height, width = left.shape[:2]

    # Create blank disparity map
    depth = np.zeros((height, width), np.uint8)

    kernel_half = int((kernel_size - 1) / 2)
    scale = 3
    max_value = 255 * 9

    for y in range(kernel_half, height-kernel_half + 1):
        for x in range(kernel_half, width-kernel_half + 1):

            # Find j where cost has minimum value
            disparity = 0
            cost_min  = 65534

            for j in range(disparity_range):
                total = 0
                value = 0

                for v in range(-kernel_half, kernel_half):
                    for u in range(-kernel_half, kernel_half):
                        value = max_value
                        if (x + u - j) >= 0:
                            value = l1_distance(int(left[y + v, x + u]),  int(
   right[y + v, (x + u) - j]))
```

```
                        total += value

                if total < cost_min:
                    cost_min = total
                    disparity = j

            # Let depth at (y, x) = j (disparity)
            # Multiply by a scale factor for visualization purpose
            depth[y, x] = disparity * scale

    if save_result == True:
        print('Saving result...')
        # Save results
        cv2.imwrite(f'window_based_l1.png', depth)
        cv2.imwrite(f'window_based_l1_color.png', cv2.applyColorMap(depth, cv2
    .COLORMAP_JET))

    print('Done.')

    return depth
```



Hình 5: Các bước tính toán giá trị disparity tại vị trí của một pixel bất kì theo phương pháp window-based matching

Tận dụng phần cài đặt gợi ý trên, các bạn sẽ thực hiện thêm một số yêu cầu sau:

- Thiết kế hai hàm có tên gọi lần lượt là:
  - **window_based_matching_l1()**
  - **window_based_matching_l2()**

Hai hàm này có chung tham số đầu vào là:

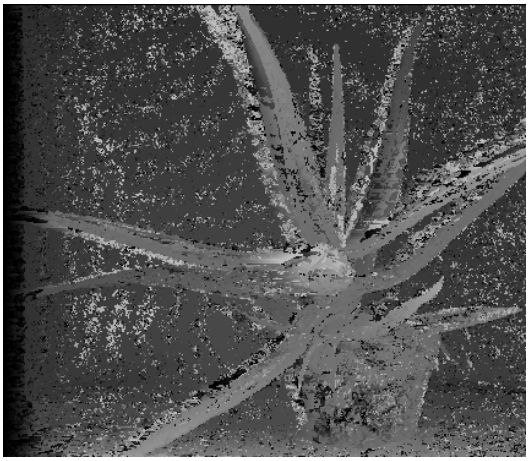
- **left_img:** đường dẫn đến ảnh chụp bên trái.
- **right_img:** đường dẫn đến ảnh chụp bên phải.
- **disparity_range:** độ dài tối đa của vùng tìm kiếm giá trị disparity tại mỗi pixel. Trong Problem 02, các bạn gán giá mặc định giá trị này = 64.
- **kernel_size:** tham số về kích thước k của window. Trong Problem 02, các bạn gán mặc định giá trị này = 3.
- **save_result:** giá trị boolean đại diện cho việc có lưu disparity map hay không? Mặc định sẽ có giá trị là True.

Cuối cùng, kết quả trả về của hàm sẽ là một numpy.ndarray đại diện cho disparity map tính được.

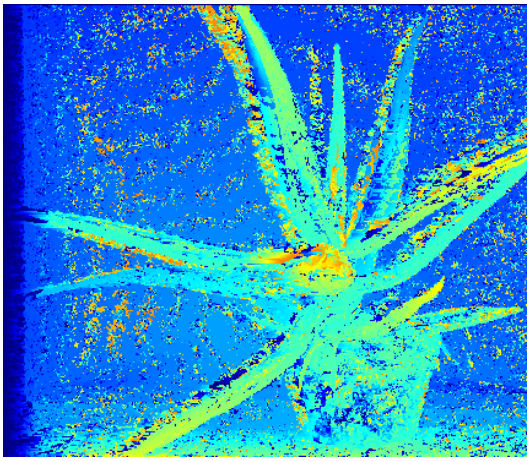
- Đối với hàm **window_based_matching_l1()**, ta dùng hàm tính cost là L1.
- Đối với hàm **window_based_matching_l2()**, ta dùng hàm tính cost là L2.
- Khi lưu kết quả, ta in ra màn hình dòng chữ **"Saving result..."** và lưu ý rằng sẽ lưu cả hai phiên bản ảnh grayscale và ảnh color map của disparity map.
- Sau khi kết thúc toàn bộ quá trình tính toán, in ra dòng chữ **"Done."**.

Kết quả minh họa khi chạy hàm **window_based_matching()** với cặp ảnh Aloe_1 sẽ có kết quả trả về như sau (các dòng comment tượng trưng cho nội dung trả về):

```
left_img_path = 'Aloe/Aloe_left_1.png'
right_img_path = 'Aloe/Aloe_right_1.png'
disparity_range = 64
kernel_size = 3

window_based_result = window_based_matching_l1(
    left_img_path,
    right_img_path,
    disparity_range,
    kernel_size=kernel_size,
    save_result=True
)

# Saving result...
# Done.

window_based_result = window_based_matching_l2(
    left_img_path,
    right_img_path,
    disparity_range,
    kernel_size=kernel_size,
    save_result=True
)

# Saving result...
# Done.
```

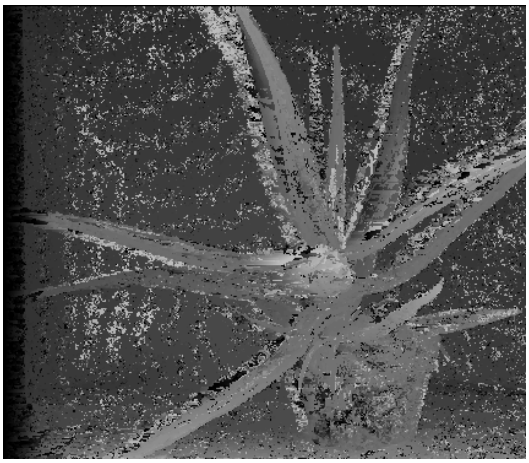


(a) Ảnh grayscale

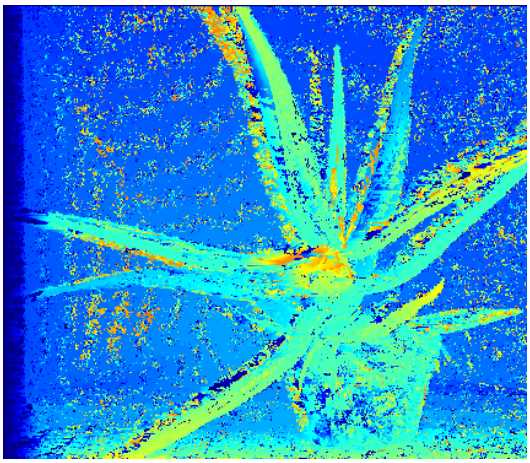


(b) Ảnh color map

Hình 6: Kết quả disparity map của phương pháp window-based matching sử dụng L1



(a) Ảnh grayscale



(b) Ảnh color map

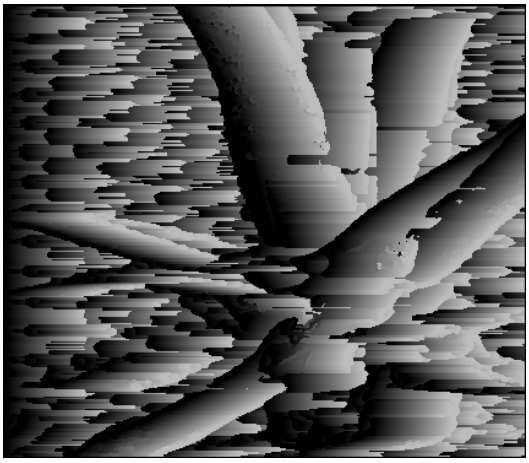
Hình 7: Kết quả disparity map của phương pháp window-based matching sử dụng L2

Problem 3: Khi sử dụng hàm tính disparity map đã xây dựng ở Problem 2 cho cặp ảnh Aloe_left_1.png và Aloe_right_2.png với tham số đầu vào disparity_range = 64 và kernel_size = 5 ở cả hai hàm cost, ta được kết quả disparity map như ảnh minh họa sau:

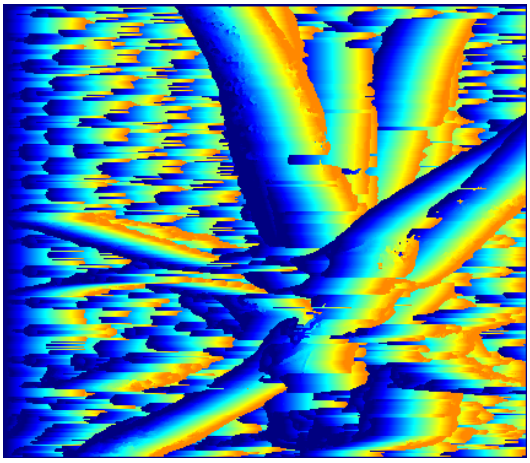
(a) Ảnh chụp bên trái

(b) Ảnh chụp bên phải

Hình 8: Ảnh stereo đầu vào của Problem 3



(a) Ảnh grayscale



(b) Ảnh color map

Hình 9: Kết quả disparity map của phương pháp window-based matching với cài đặt disparity_range = 64 và kernel_size = 5

Có thể thấy với sự thay đổi của các giá trị tham số đầu vào như trên, kết quả disparity map đã phần nào tệ đi (bị nhiễu). Các bạn hãy sử dụng code ở Problem 2 để tạo ra ảnh disparity map với cài đặt này và giải thích (sử dụng markdown) vì sao lại xảy ra kết quả như vậy.

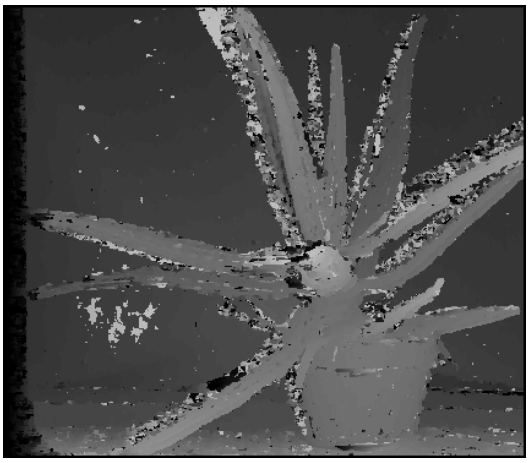
Problem 4: Dựa trên hàm tính disparity map theo phương thức window-based matching ở Problem 2 và coi các window là các vector, hãy cài đặt Cosine Similarity trong việc tính sự tương quan giữa hai pixel ảnh trái phải để giải quyết vấn đề ở Problem 3. Công thức Cosine Similarity được mô tả như sau:

$$cosine\_similarity(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\|\|\vec{y}\|} = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}$$

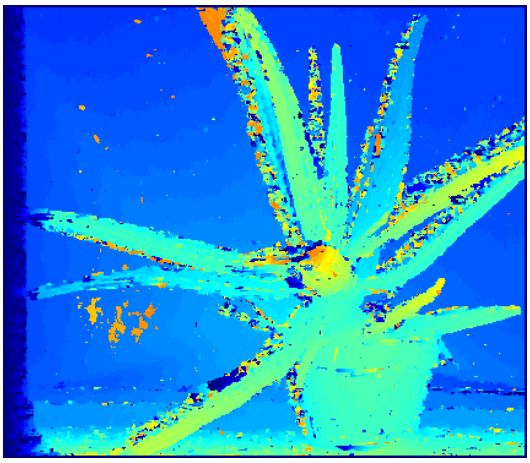
Như vậy, phần cài đặt thuật toán gần như tương tự với phiên bản ở Problem 02. Tại đây, ta sẽ thay đổi lại một chút cách sử dụng giá trị của các window để có thể đưa vào công thức cosine similarity. Bằng cách coi cửa sổ là vector, phần cài đặt code sẽ thay đổi lại như sau:

```python
import cv2
import numpy as np

def cosine_similarity(x, y):
    numerator = np.dot(x, y)
    denominator = np.linalg.norm(x) * np.linalg.norm(y)

    return numerator / denominator

def window_based_matching(left_img, right_img, disparity_range, kernel_size=5, save_result=True):
    # Read left, right images then convert to grayscale
    left  = cv2.imread(left_img, 0)
    right = cv2.imread(right_img, 0)

    left  = left.astype(np.float32)
    right = right.astype(np.float32)

    height, width = left.shape[:2]

    # Create blank disparity map
    depth = np.zeros((height, width), np.uint8)
    kernel_half = int((kernel_size - 1) / 2)
    scale = 3

    for y in range(kernel_half, height-kernel_half):
        for x in range(kernel_half, width-kernel_half):

            # # Find j where cost has minimum value
            disparity = 0
            cost_optimal  = -1

            for j in range(disparity_range):
                d = x - j
                cost = -1
                if (d - kernel_half) > 0:
                    wp = left[(y-kernel_half):(y+kernel_half)+1, (x-kernel_half):(x+kernel_half)+1]
                    wqd = right[(y-kernel_half):(y+kernel_half)+1, (d-kernel_half):(d+kernel_half)+1]

                    wp_flattened = wp.flatten()
                    wqd_flattened = wqd.flatten()

                    cost = cosine_similarity(wp_flattened, wqd_flattened)

                if cost > cost_optimal:
                    cost_optimal = cost
                    disparity = j

                # Let depth at (y, x) = j (disparity)
```

```
                # Multiply by a scale factor for visualization purpose
                depth[y, x] = disparity * scale

    if save_result == True:
        print('Saving result...')
        # Save results
        cv2.imwrite('window_based_cosine_similarity.png', depth)
        cv2.imwrite('window_based_cosine_similarity_color.png', cv2.
    applyColorMap(depth, cv2.COLORMAP_JET))

    print('Done.')

    return depth
```



(a) Ảnh grayscale



(b) Ảnh color map

Hình 10: Kết quả disparity map của phương pháp window-based matching sử dụng Cosine Similarity với cài đặt tham số và cặp ảnh đầu vào ở Problem 3

III. Trắc nghiệm

1. Ứng dụng nào sau đây có thể liên quan đến việc sử dụng disparity map?
   (*a*) Object Detection.
   (*b*) Image Classification.
   (*c*) Text Retrieval.
   (*d*) Depth Estimation for 3D Reconstruction.

2. Lý do nào sau đây là lợi ích trong việc sử dụng phương pháp window-based matching so với phương pháp pixel-wise matching?
   (*a*) Tốc độ tính toán và xử lý nhanh hơn.
   (*b*) Hoạt động tốt trên độ phân giải ảnh thấp.
   (*c*) Kết quả disparity map mượt hơn.
   (*d*) Loại bỏ được bước Calibration trong hệ thống.

3. Câu nào sau đây mô tả một điểm yếu của phương pháp pixel-wise matching?
   (*a*) Yêu cầu ảnh có độ phân giải cao.
   (*b*) Độ phức tạp thuật toán cao.
   (*c*) Kết quả disparity map bị nhiễu cao.
   (*d*) Nhạy cảm với điều kiện ánh sáng.

4. Vai trò của Disparity Range là gì?
   (*a*) Kích thước của search window.
   (*b*) Vùng tìm kiếm giá trị disparity.
   (*c*) Định nghĩa hàm cost.
   (*d*) Ảnh hưởng đến độ sáng của disparity map.

5. Trong phương pháp window-based matching, kích thước window sẽ ảnh hưởng đến điều gì?
   (*a*) Độ chính xác của disparity map.
   (*b*) Độ sáng của disparity map.
   (*c*) Tốc độ của thuật toán.
   (*d*) Độ sâu của disparity map.

6. Lý do nào sau đây giải thích việc nhân disparity value với tỉ số $\frac{255}{D}$?
   (*a*) Cải thiện độ phân giải.
   (*b*) Tăng cường độ chính xác.
   (*c*) Tăng độ sáng ảnh.
   (*d*) Dùng trong việc visualization kết quả.

- Hết -