AI VIETNAM
All-in-One Course
(TA Session)

Image Depth Estimation

Project

Dinh-Thang Duong – TA

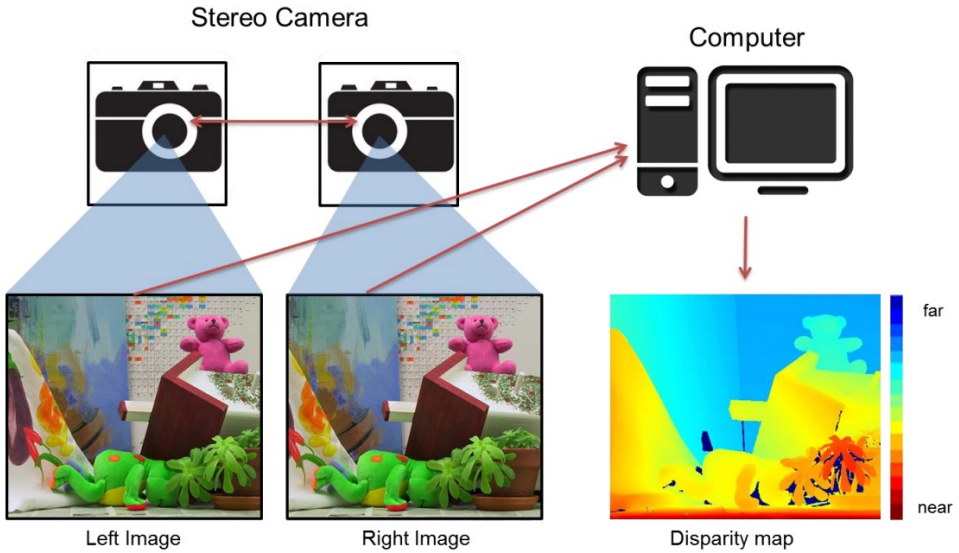
Year 2024

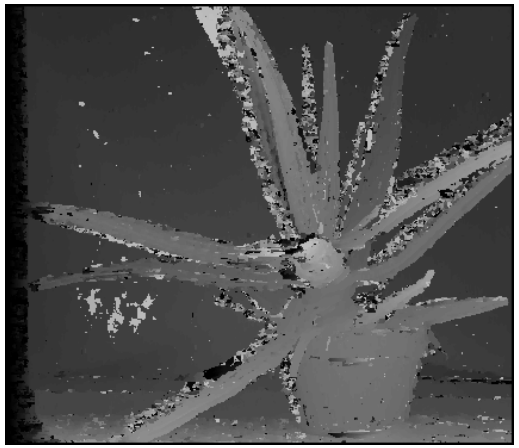
Outline

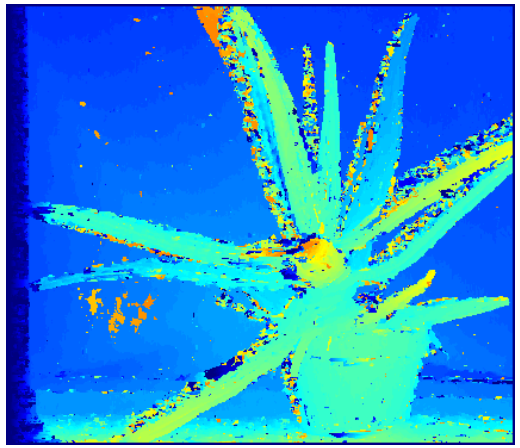
- Introduction
- Problem 01
- Problem 02
- Problem 03
- Problem 04
- Question

Getting Started

❖ Objectives







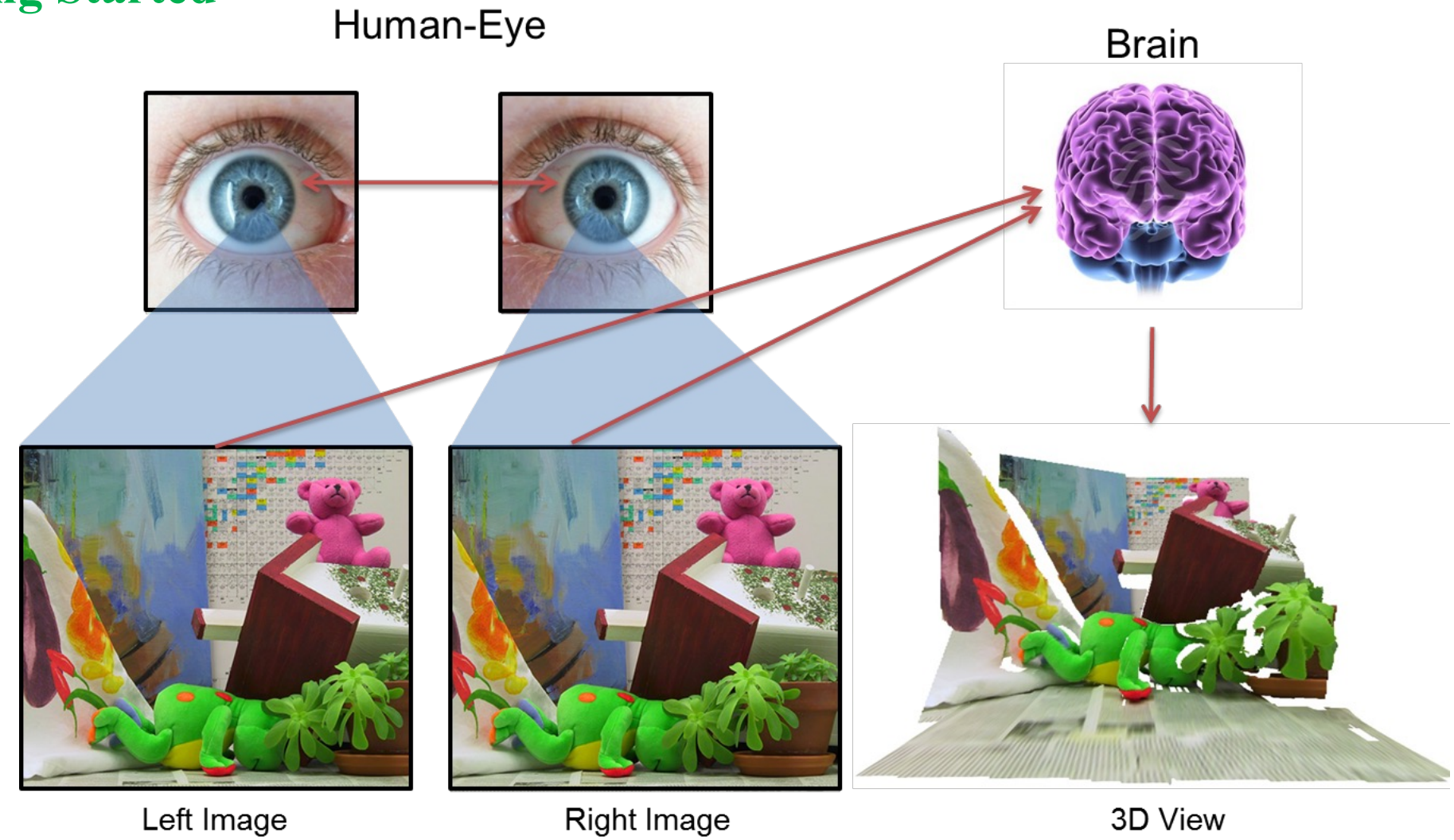
Our objectives:

- Introduction to Stereo Matching.
- Implement algorithms for calculating disparity map:
  - Pixel-wise matching.
  - Window-based matching.
- Investigate the output of each algorithm through visualization.

Introduction

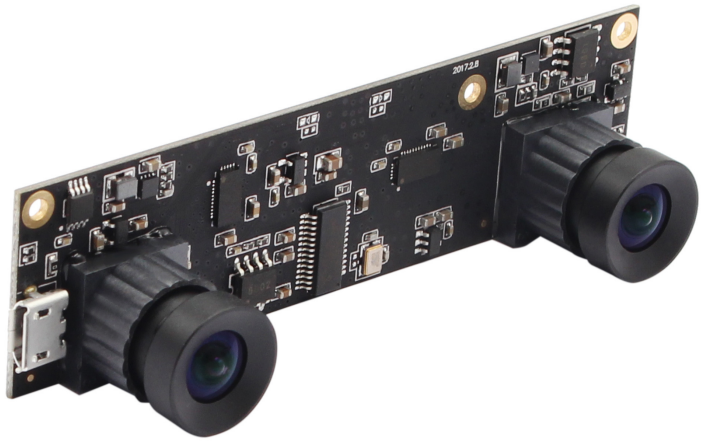
Introduction

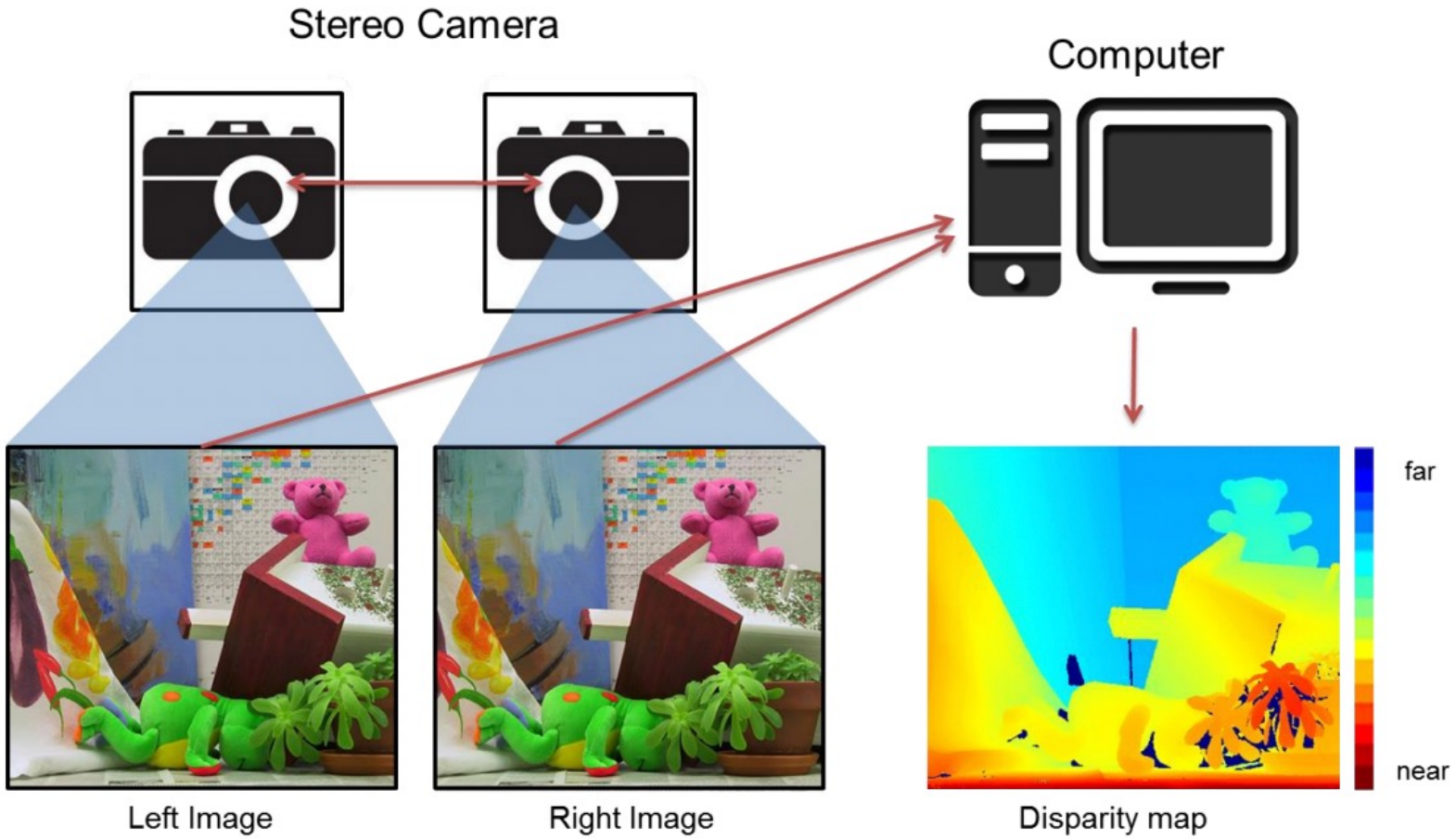
❖ Getting Started



Introduction

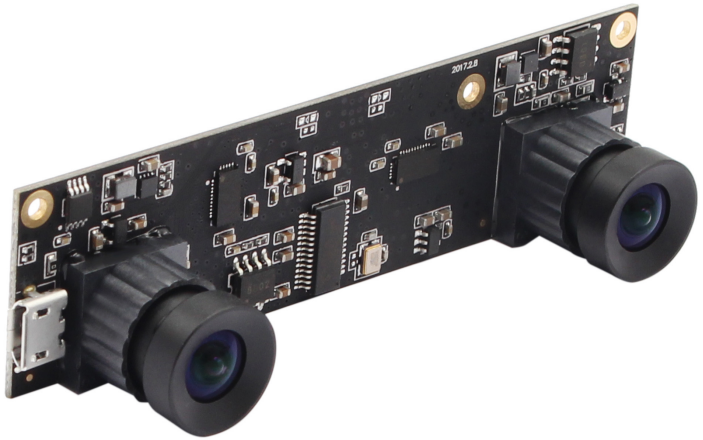
❖ Getting Started

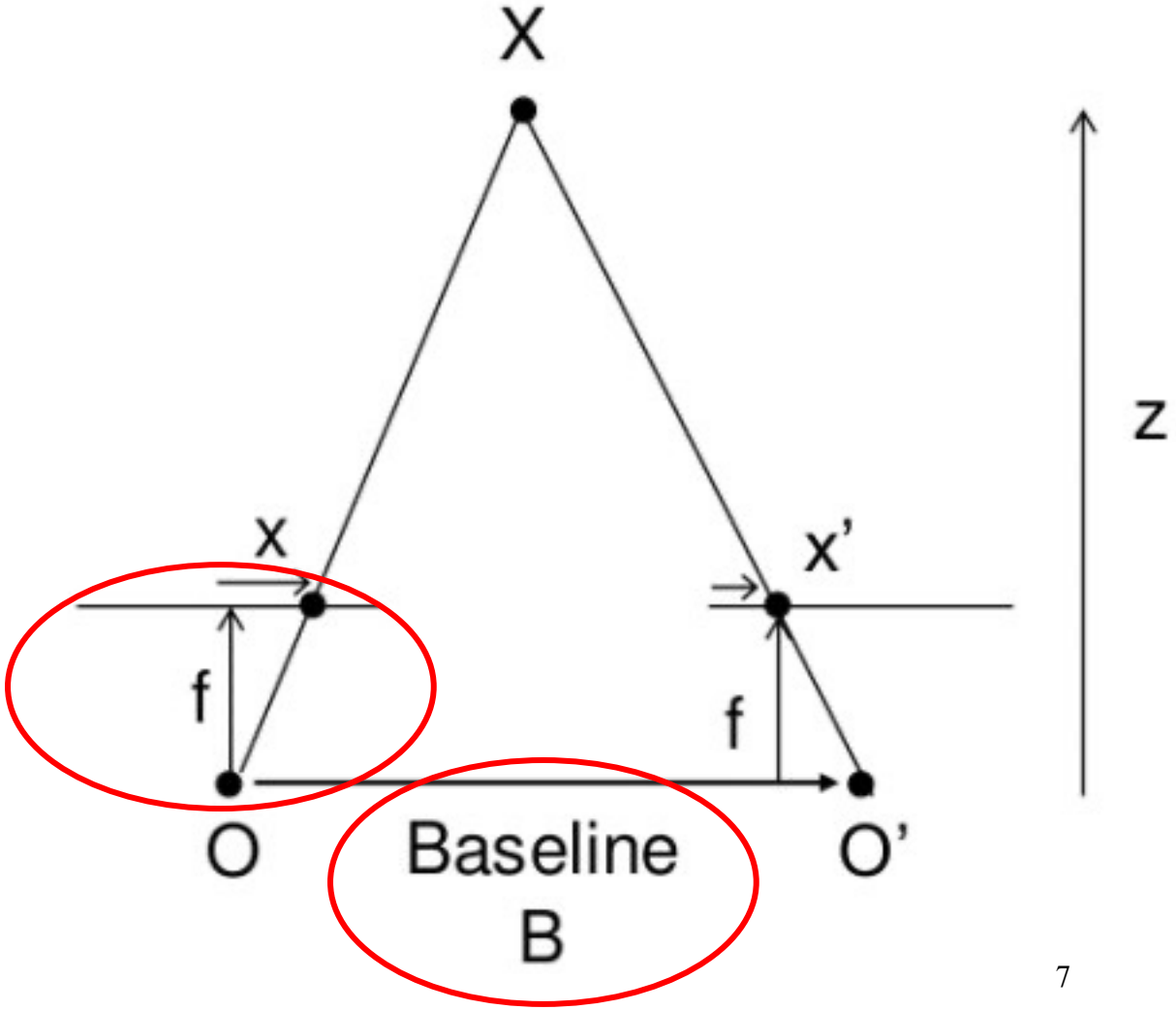




Introduction

❖ Stereo Camera

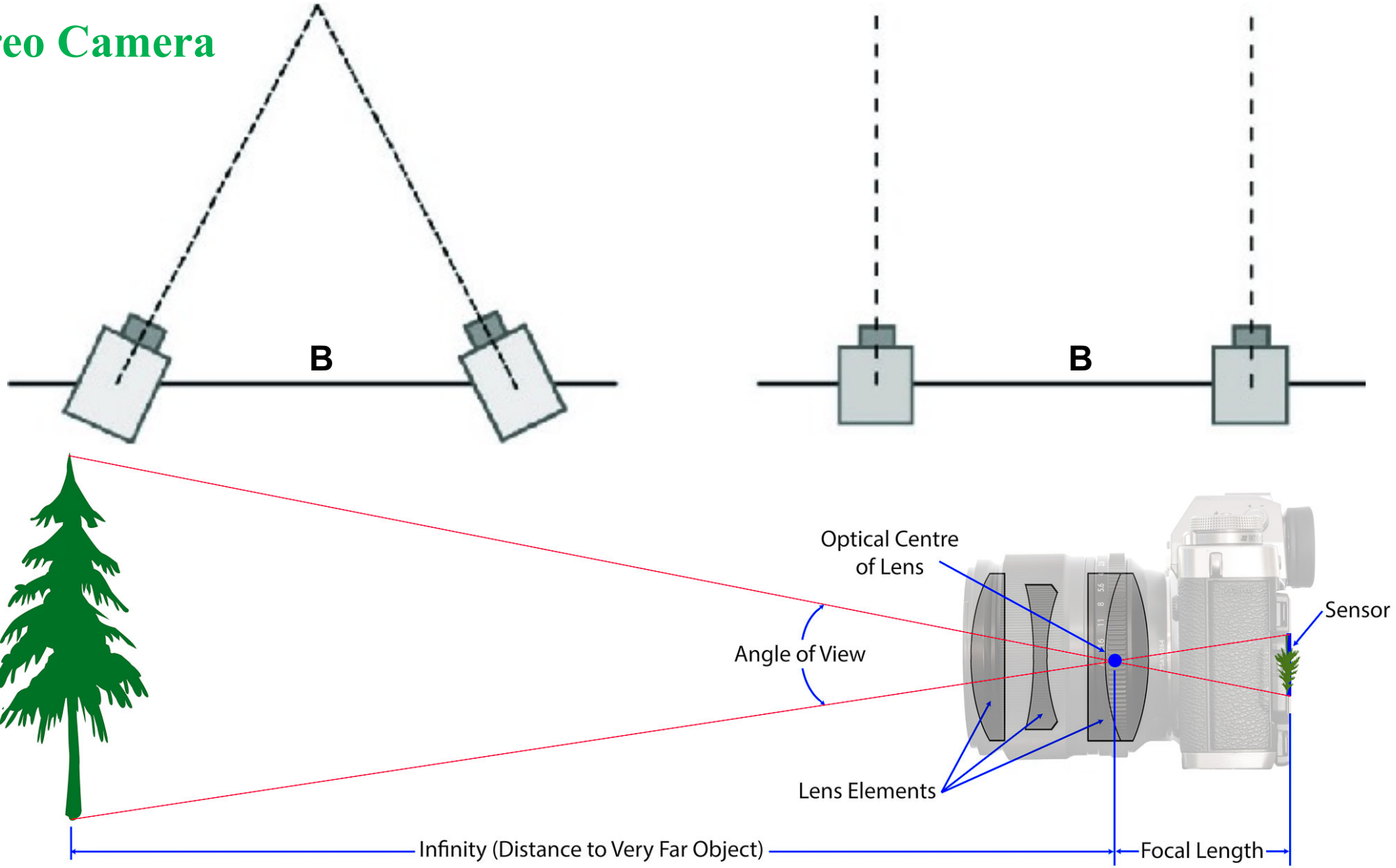




https://docs.opencv.org/4.x/stereo_depth.jpg

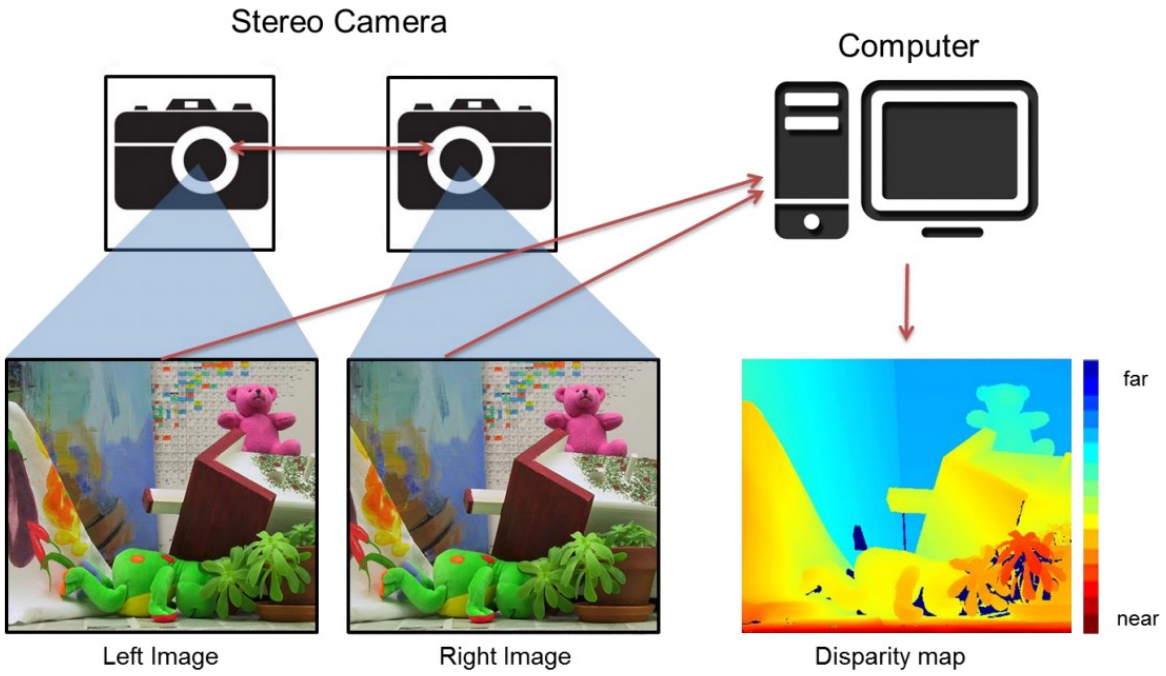
Introduction

❖ Stereo Camera



Introduction

❖ Stereo Matching



Stereo Matching: A computer vision technique used to determine the depth of objects in a scene by comparing two or more images taken from slightly different viewpoints. By analyzing the disparity between corresponding points in these images, stereo matching algorithms estimate the distance to various points, creating a 3D representation of the scene.

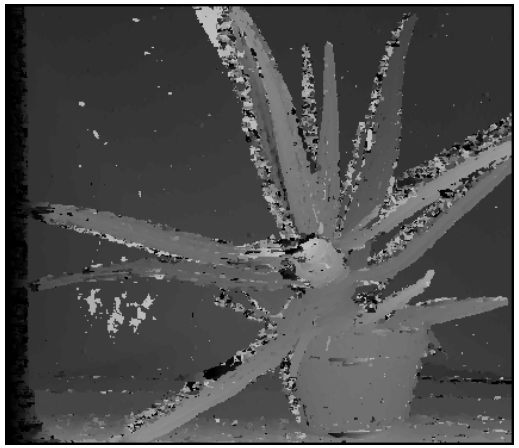
Introduction

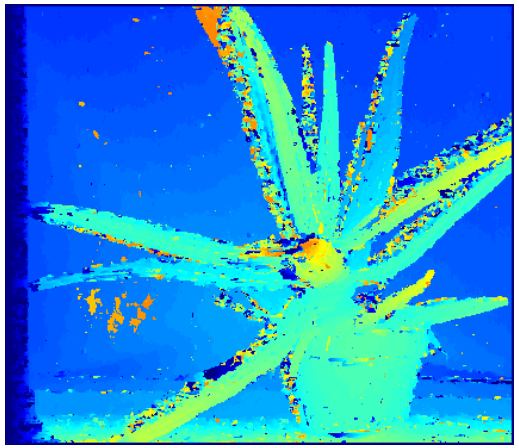
❖ Stereo Matching

Left Image

Right Image

stereo_matching()

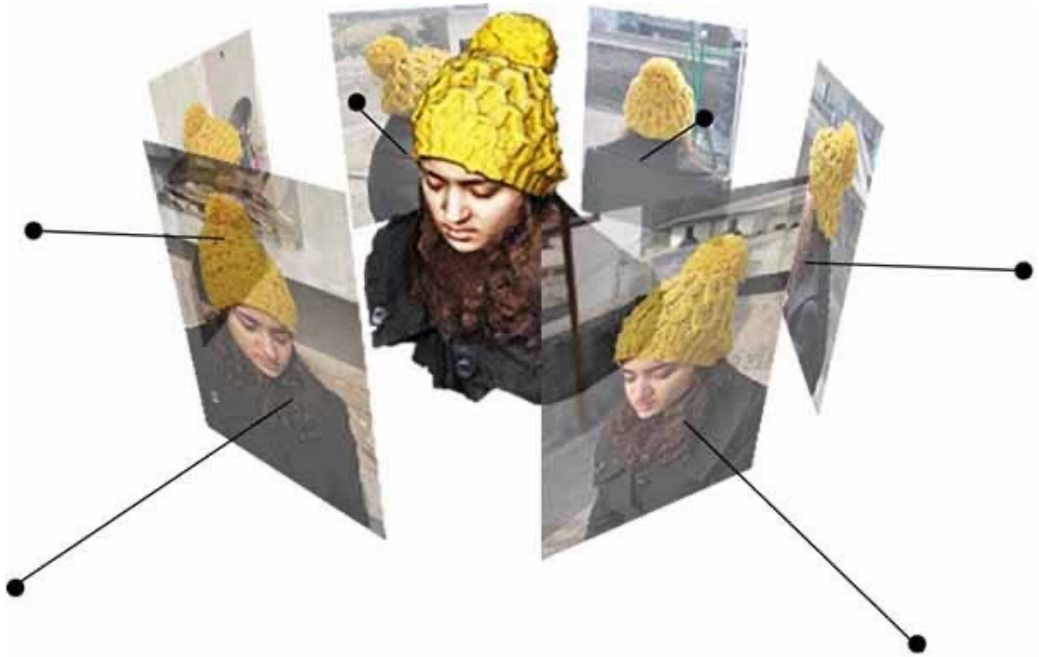




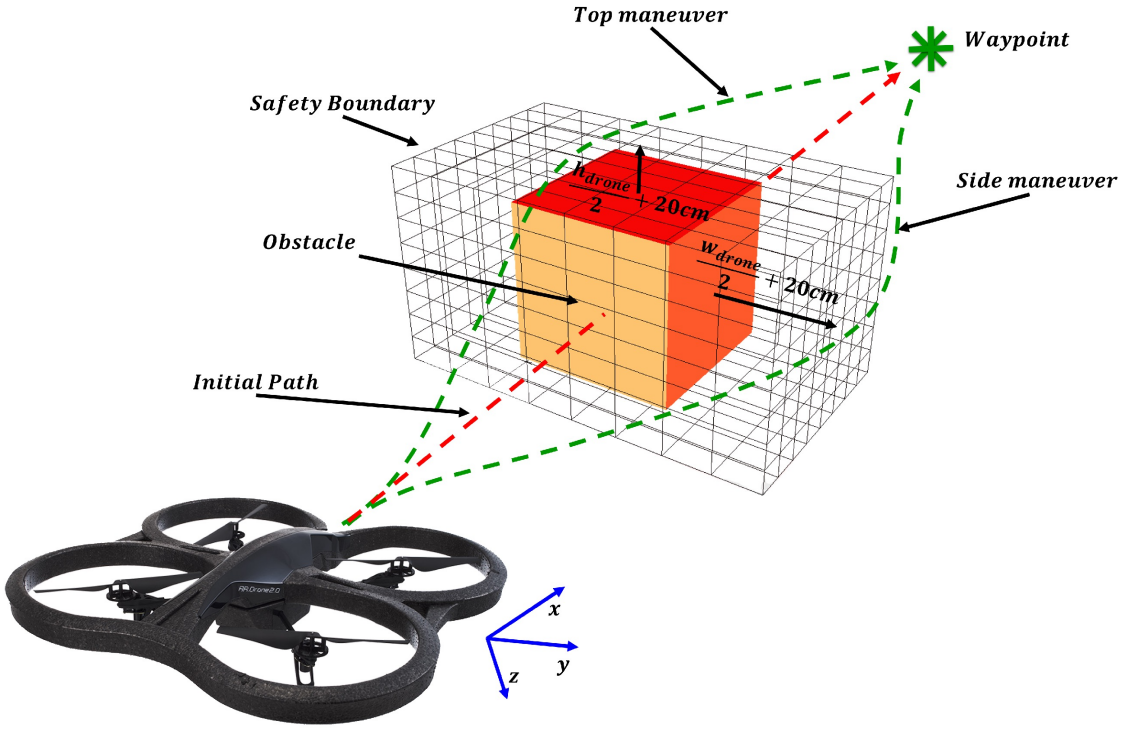
Disparity Map

Introduction

❖ Stereo Matching Applications



3D Reconstruction and Modeling



Obstracle Detection and Avoidance

Introduction

❖ Stereo Matching Challenges

Normal stereo pair | ADCensus | SAD

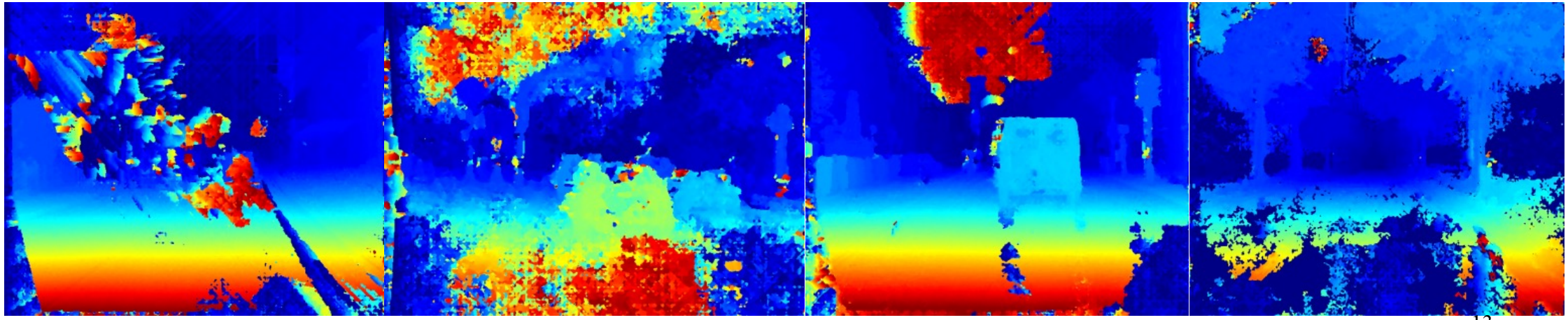
Stereo pair with different illumination | ADCensus | SAD

Introduction

❖ Stereo Matching Challenges

Left and right images

Estimated disparity maps

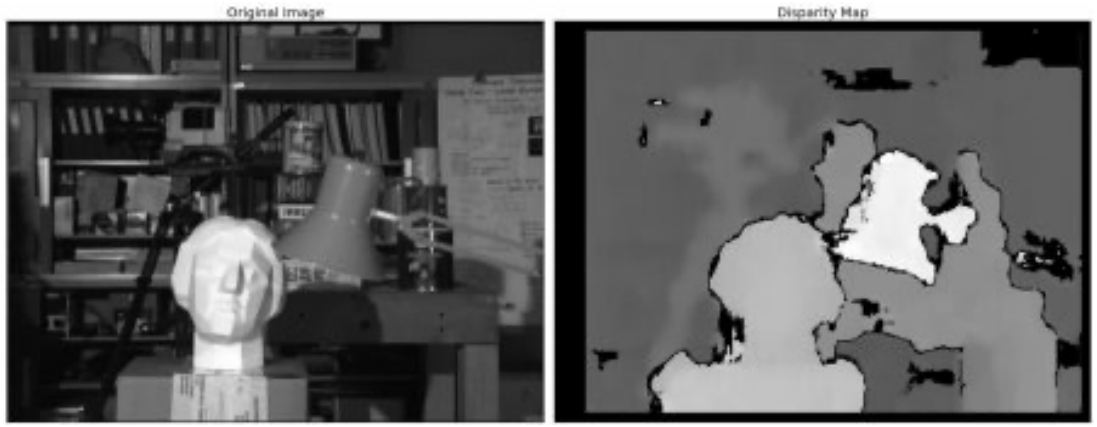


Sun Flare | City at Night | Rain Flare | Rain Blur

Introduction

❖ Project Objective

```python
import numpy as np
import cv2

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

left = cv2.imread(left_img_path, cv2.IMREAD_GRAYSCALE)
right = cv2.imread(right_img_path, cv2.IMREAD_GRAYSCALE)

stereo = cv2.StereoBM_create(
    numDisparities=disparity_range,
    blockSize=15
)
disparity = stereo.compute(left, right)
cv2_imshow(disparity)
```

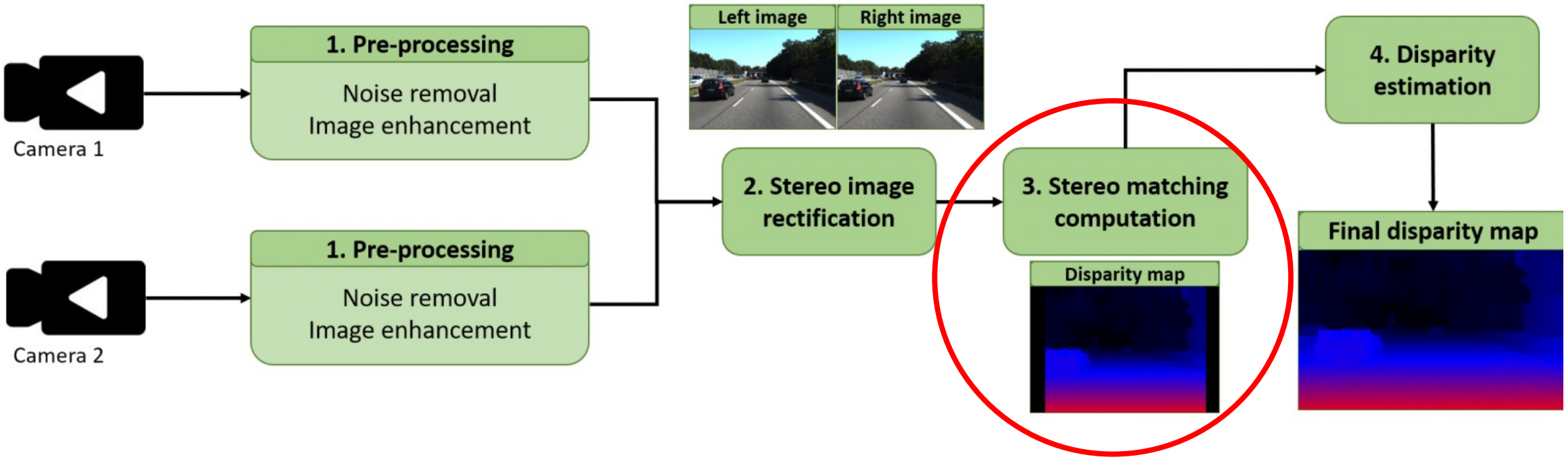


Implement basic Disparity Map calculation functions

https://docs.opencv.org/4.x/dd/d53/tutorial_py_depthmap.htmlti

Introduction

❖ Stereo Matching Pipeline



Problem 01

Problem 01

❖ Problem Statement

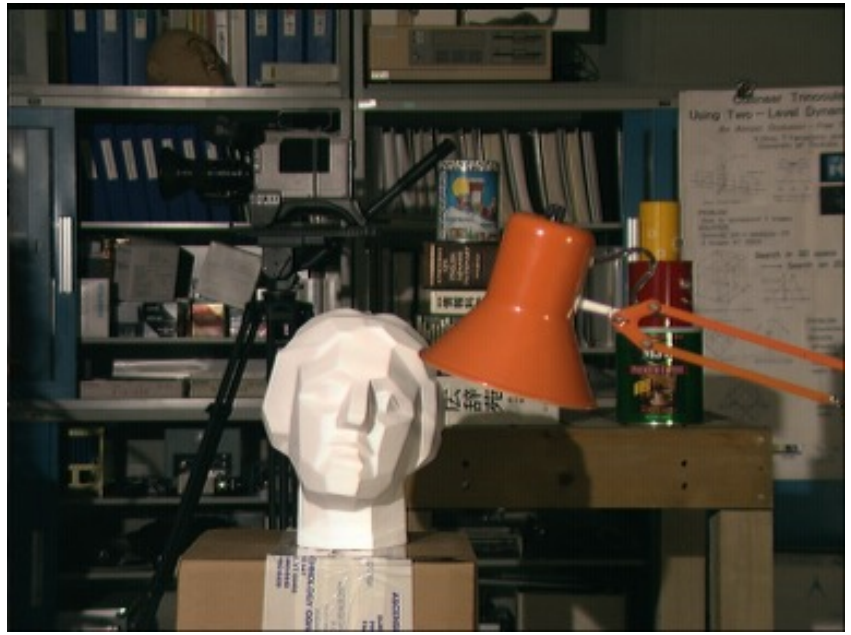
Build a function to calculate the disparity map of two input stereo images (left image (L) and right image (R)) using pixel-wise matching method. The calculation steps in this method can be described through the following steps:

1. Read the left (left) and right (right) images as grayscale images and convert them to np.float32.
2. Initialize two variables height, width with values equal to the height and width of the left image.
3. Initialize a zero-zero matrix (depth) with shape equal to (height, width).
4. For each pixel at position (h, w) (browsing from left to right, top to bottom) perform the following steps:
   a) Calculate the cost (L1 or L2) between the pairs of pixels $left[h, w]$ and $right[h, w - d]$ (where $d \in [0, disparity\_range]$) (In this problem, $disparity\_range = 16$) . If $(w - d) < 0$, then assign the value $cost = \max\_cost$ ($\max\_cost = 255$ if using L1 or $\max\_cost = 255^2$ if using L2).
   b) With the list of calculated costs, choose the d ($d_{optimal}$) value where the cost is the smallest.
   c) Assign $depth[h, w] = d_{optimal} \times scale$. (In this problem, $scale = 16$).

Problem 01

❖ Problem Statement

In this problem, we run on **Tsukuba** stereo images. You can download from here.



Left Image

Right Image

Problem 01

❖ Method: Pixel-wise Matching

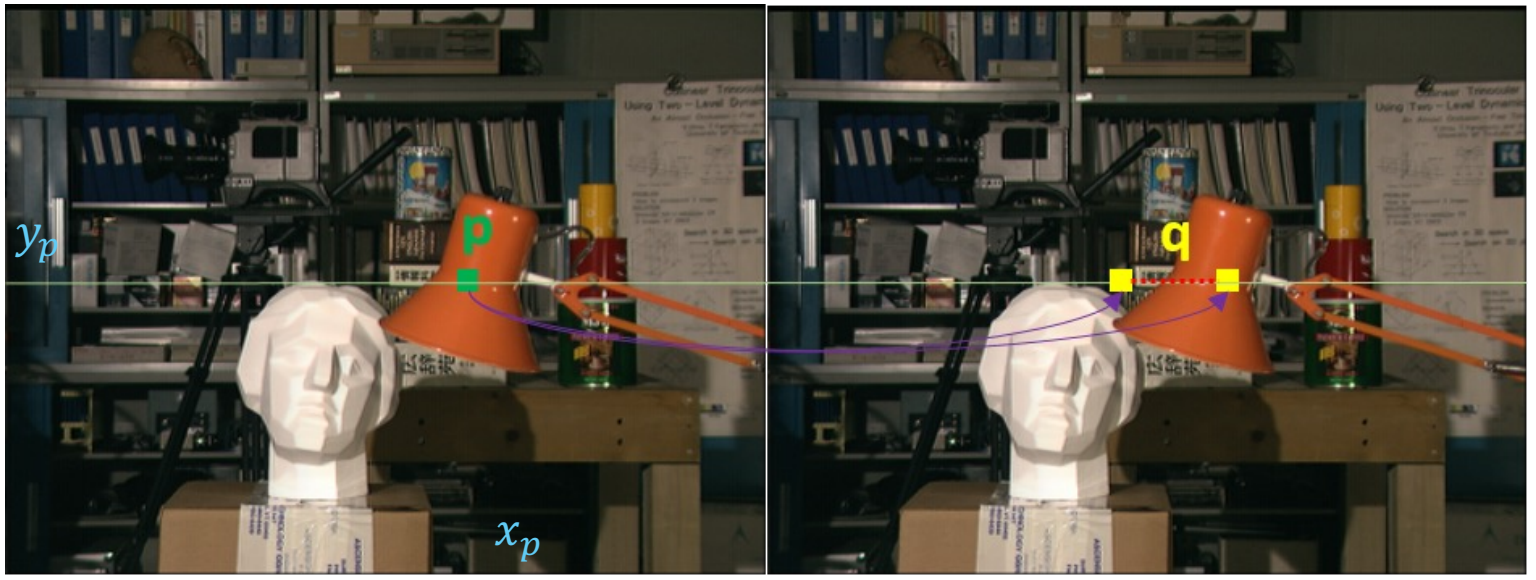
L is the left image

R is the right image

$L(\boldsymbol{p})$ is the (vector) value of $\mathbf{p}$

$$\boldsymbol{p} = \begin{bmatrix} x_p \\ y_p \end{bmatrix} \qquad d \in D$$

$$\boldsymbol{d} = \begin{bmatrix} d \\ 0 \end{bmatrix} \qquad C_1(\boldsymbol{p}, \boldsymbol{d}) = |L(\boldsymbol{p}) - R(\boldsymbol{p} - \boldsymbol{d})|$$



Left Image Right Image

Finding d so that $C_1(\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{d})$ is minimum.

$$d = \underset{d \in D}{\operatorname{argmin}}\left(C_1(\boldsymbol{p}, \boldsymbol{d})\right)$$

Then, d is the value for the pixel $\mathbf{p}$ in disparity map

AI VIETNAM
All-in-One Course
(TA Session)

Problem 01

❖ Method: Pixel-wise Matching

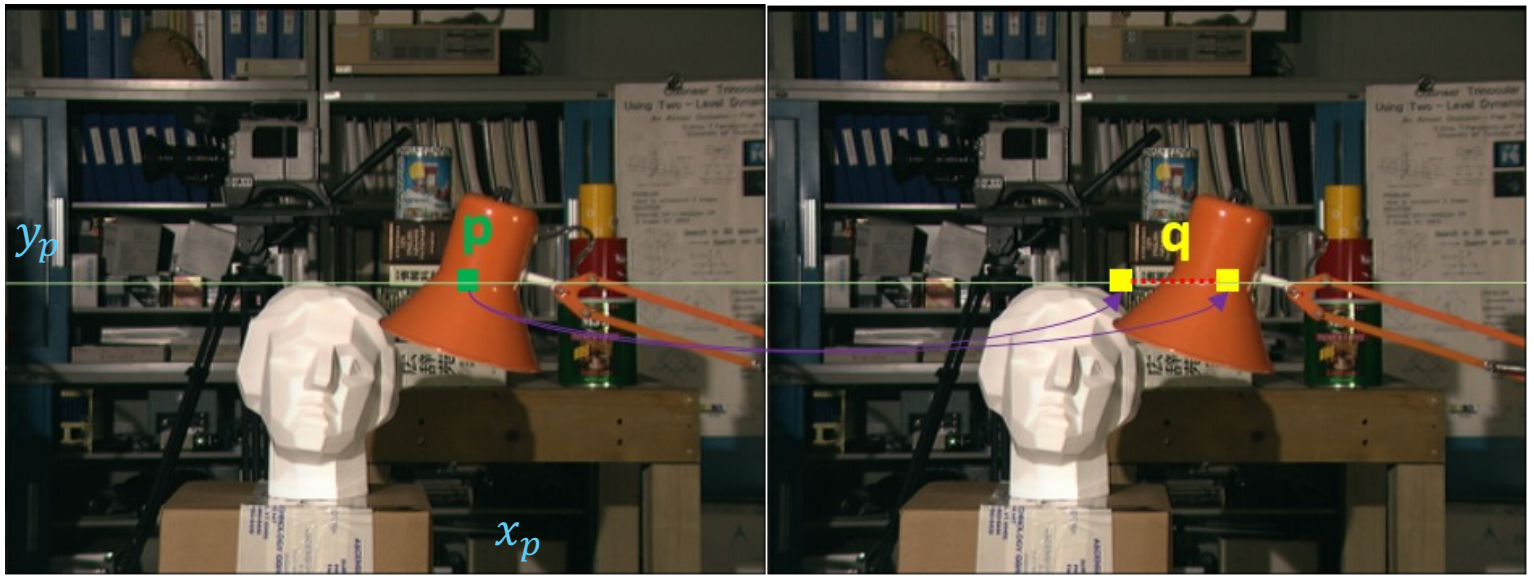
L is the left image

R is the right image

$L(\boldsymbol{p})$ is the (vector) value of **p**

$$\boldsymbol{p} = \begin{bmatrix} x_p \\ y_p \end{bmatrix} = \begin{bmatrix} 234 \\ 140 \end{bmatrix}$$

$$D = 16$$



Left Image

Right Image

q

$$\begin{bmatrix} x_p - 0 \\ y_p \end{bmatrix} = \begin{bmatrix} 234 \\ 140 \end{bmatrix} \quad \begin{bmatrix} x_p - 8 \\ y_p \end{bmatrix} = \begin{bmatrix} 226 \\ 140 \end{bmatrix}$$

$$\begin{bmatrix} x_p - 1 \\ y_p \end{bmatrix} = \begin{bmatrix} 233 \\ 140 \end{bmatrix} \quad \begin{bmatrix} x_p - 9 \\ y_p \end{bmatrix} = \begin{bmatrix} 225 \\ 140 \end{bmatrix}$$

$$\begin{bmatrix} x_p - 2 \\ y_p \end{bmatrix} = \begin{bmatrix} 232 \\ 140 \end{bmatrix} \quad \begin{bmatrix} x_p - 10 \\ y_p \end{bmatrix} = \begin{bmatrix} 224 \\ 140 \end{bmatrix}$$

$$\begin{bmatrix} x_p - 3 \\ y_p \end{bmatrix} = \begin{bmatrix} 231 \\ 140 \end{bmatrix} \quad \begin{bmatrix} x_p - 11 \\ y_p \end{bmatrix} = \begin{bmatrix} 223 \\ 140 \end{bmatrix}$$

$$\begin{bmatrix} x_p - 4 \\ y_p \end{bmatrix} = \begin{bmatrix} 230 \\ 140 \end{bmatrix} \quad \begin{bmatrix} x_p - 12 \\ y_p \end{bmatrix} = \begin{bmatrix} 222 \\ 140 \end{bmatrix}$$

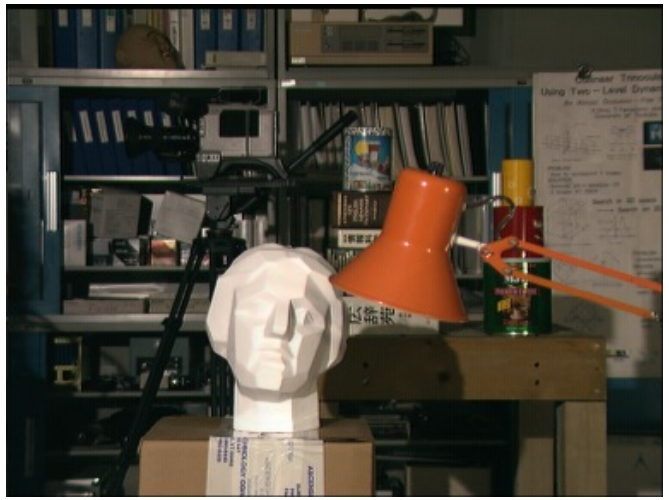
$$\begin{bmatrix} x_p - 5 \\ y_p \end{bmatrix} = \begin{bmatrix} 229 \\ 140 \end{bmatrix} \quad \begin{bmatrix} x_p - 13 \\ y_p \end{bmatrix} = \begin{bmatrix} 221 \\ 140 \end{bmatrix}$$

$$\begin{bmatrix} x_p - 6 \\ y_p \end{bmatrix} = \begin{bmatrix} 228 \\ 140 \end{bmatrix} \quad \begin{bmatrix} x_p - 14 \\ y_p \end{bmatrix} = \begin{bmatrix} 220 \\ 140 \end{bmatrix}$$

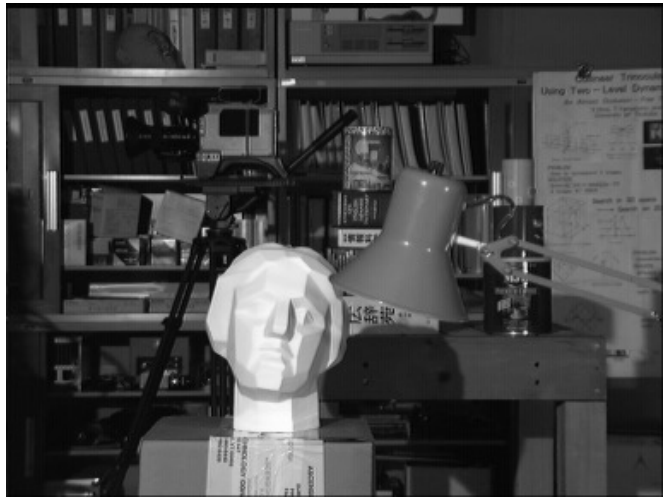
$$\begin{bmatrix} x_p - 7 \\ y_p \end{bmatrix} = \begin{bmatrix} 227 \\ 140 \end{bmatrix} \quad \begin{bmatrix} x_p - 15 \\ y_p \end{bmatrix} = \begin{bmatrix} 219 \\ 140 \end{bmatrix}$$

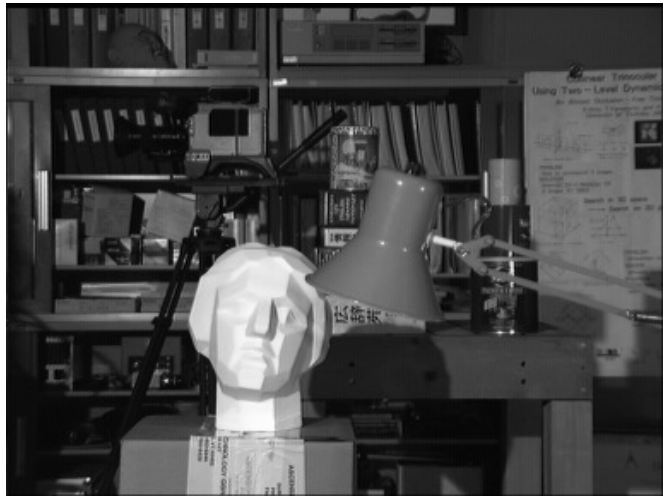
Problem 01

❖ Step 1: Type Conversion



To grayscale





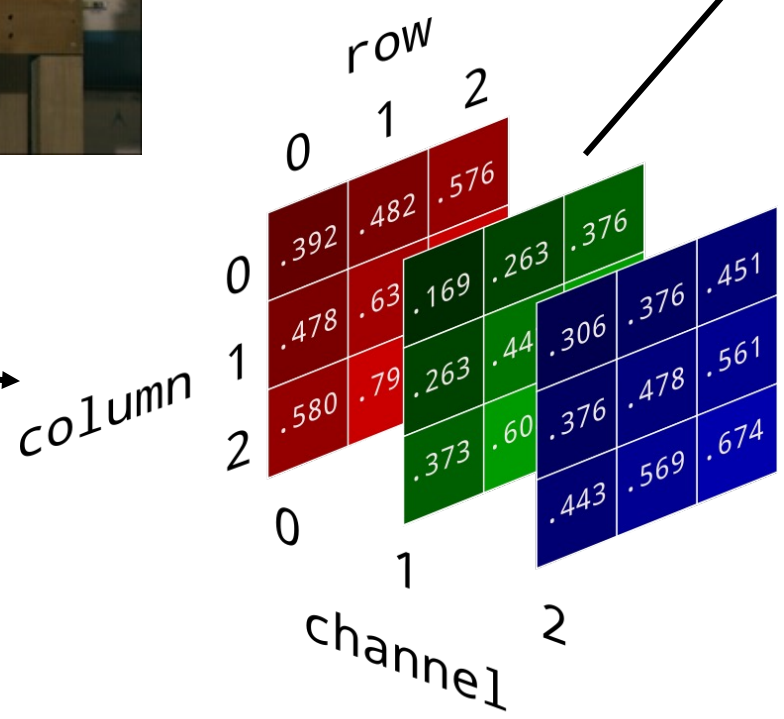
Type conversion

np.uint8 to np.float32

Problem 01

❖ Step 1: Type Conversion (Convert to grayscale)

$$gray[x, y] = 0.299[x, y, 0] + 0.587[x, y, 1] + 0.114[x, y, 2]$$



1	2	2	3	2	2
2	2	3	1	2	

Grayscale image (visualized as array)

https://e2eml.school/convert_rgb_to_grayscale

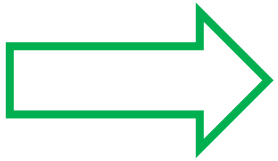
Problem 01

❖ Step 1: Type Conversion

```
[[ 1  2  2 ...  1  2  1]
 [ 2  4  3 ...  2  4  2]
 [ 2  4  3 ... 21 23 11]
 ...
 [28 14  9 ... 76 76 37]
 [34 14 10 ... 88 87 42]
 [19  6  4 ... 47 46 22]]
```

Tsukuba left image array in np.uint8 (left)

`left - 200`



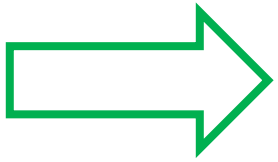
```
[[ 57  58  58 ...  57  58  57]
 [ 58  60  59 ...  58  60  58]
 [ 58  60  59 ...  77  79  67]
 ...
 [ 84  70  65 ... 132 132  93]
 [ 90  70  66 ... 144 143  98]
 [ 75  62  60 ... 103 102  78]]
```

After subtract by 200

```
[[ 1.  2.  2. ...  1.  2.  1.]
 [ 2.  4.  3. ...  2.  4.  2.]
 [ 2.  4.  3. ... 21. 23. 11.]
 ...
 [28. 14.  9. ... 76. 76. 37.]
 [34. 14. 10. ... 88. 87. 42.]
 [19.  6.  4. ... 47. 46. 22.]]
```

Tsukuba left image array in np.float32 (left)

`left - 200`

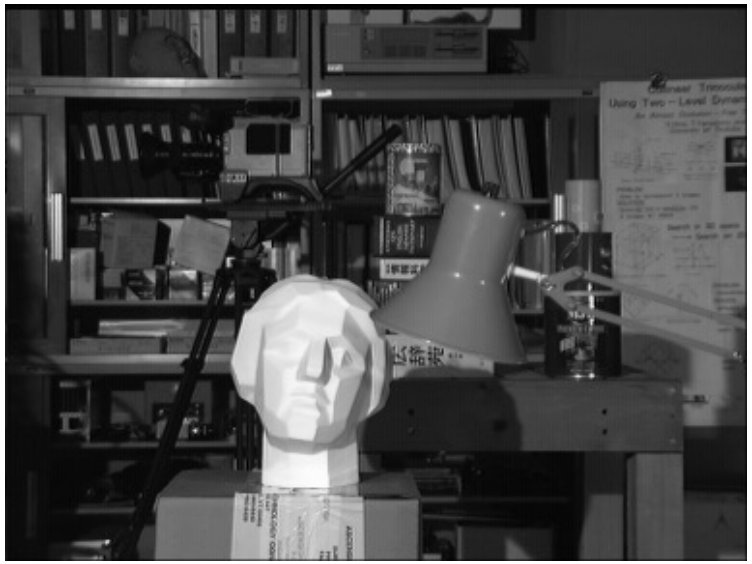


```
[[-199. -198. -198. ... -199. -198. -199.]
 [-198. -196. -197. ... -198. -196. -198.]
 [-198. -196. -197. ... -179. -177. -189.]
 ...
 [-172. -186. -191. ... -124. -124. -163.]
 [-166. -186. -190. ... -112. -113. -158.]
 [-181. -194. -196. ... -153. -154. -178.]]
```

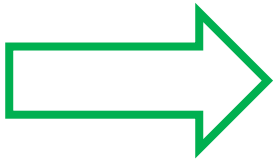
After subtract by 200

Problem 01

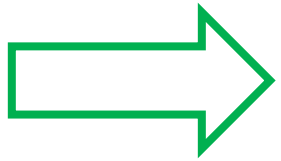
❖ Step 2: Zeros Matrix



Get height, width



height: 288
width: 384



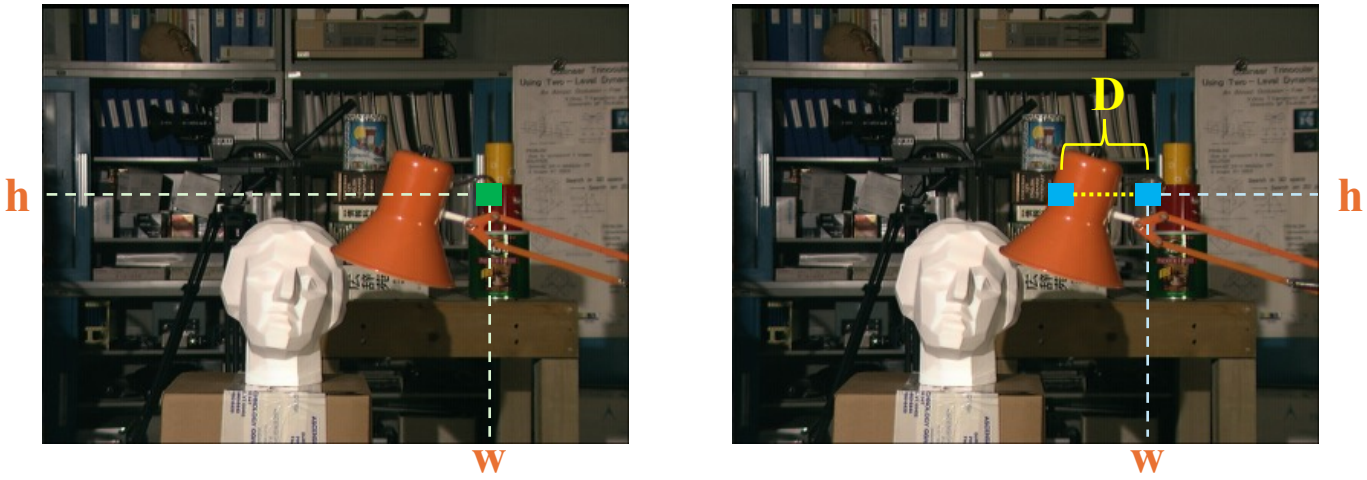
```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

Create zero matrix with height, width in np.uint8 (depth)

Problem 01

❖ Step 3: Depth Calculation

❖ **For each (h, w) point:**



Left image (L) Right image (R)

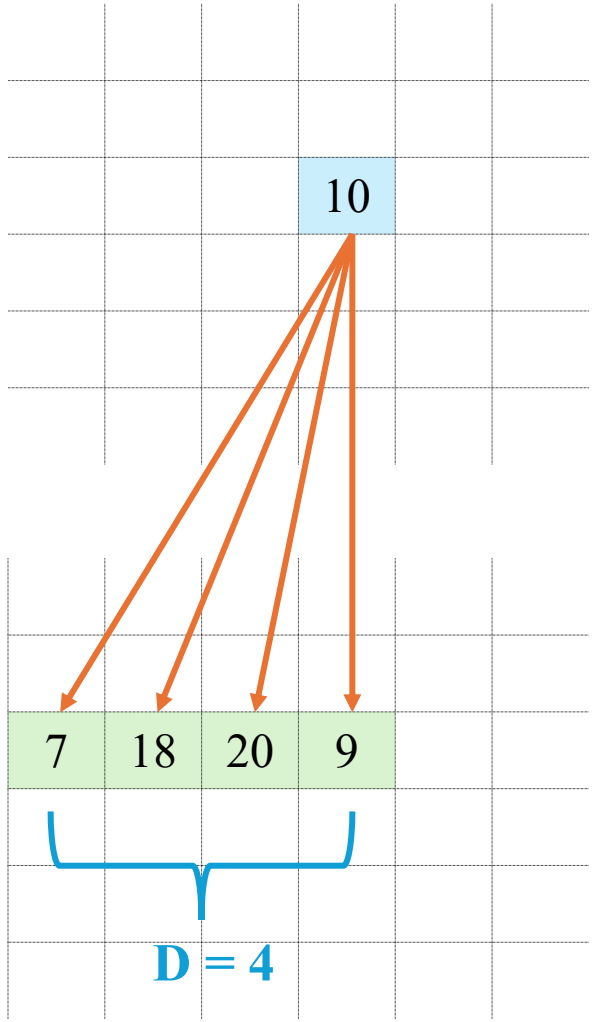
L(h, w), R(h, w)

$$disparity = \underset{d}{arg\min}(cost(L[h,w], R[h,w - d])\ \{d \in [0, D]\}$$

$$depth[h,w] = disparity \times scale$$

Problem 01

❖ Step 3: Depth Calculation



$$disparity = \underset{d}{argmin}(cost(L[h,w], R[h,w-d])\ \{d \in [0, D]\}$$

$$L1 = cost(x,y) = |x-y|$$

$$L2 = cost(x,y) = (x-y)^2$$

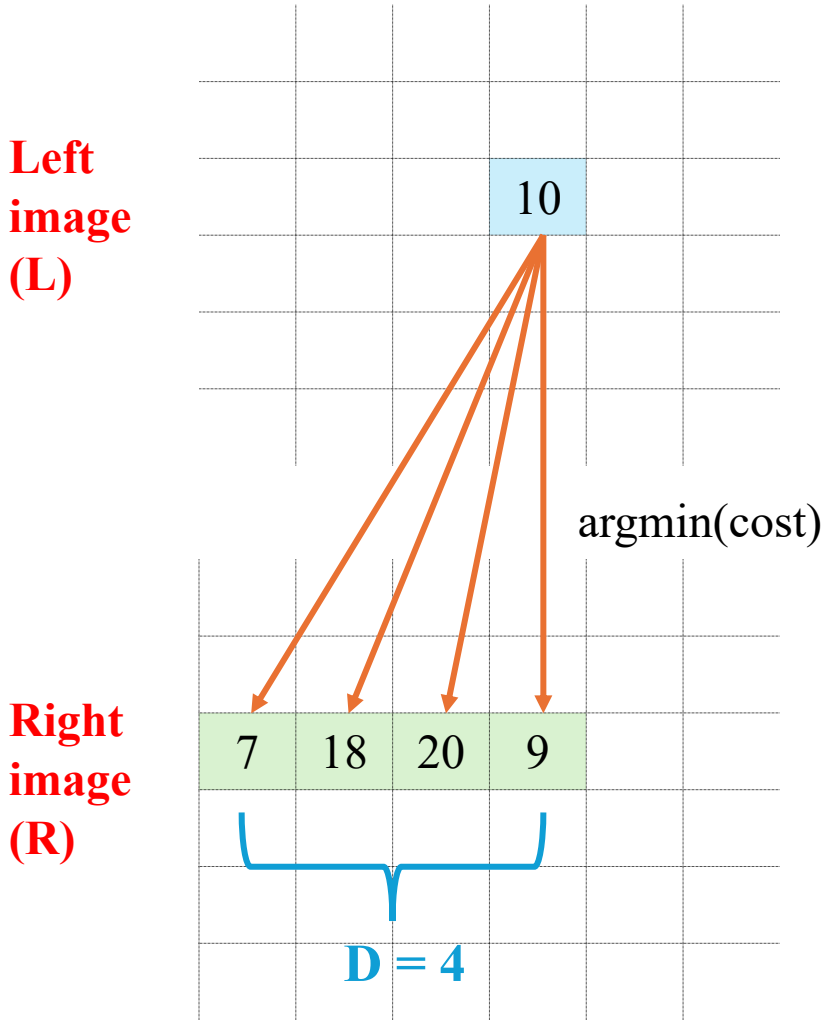
- distance_1 = cost(left(3, 4), right(3, 4))= 1
- distance_2 = cost(left(3, 4), right(3, 3)) = 10
- distance_3 = cost(left(3, 4), right(3, 2)) = 8
- distance_4 = cost(left(3, 4), right(3, 1)) = 3

- distance_1 = cost(left(3, 4), right(3, 4))= 1
- distance_2 = cost(left(3, 4), right(3, 3)) = 100
- distance_3 = cost(left(3, 4), right(3, 2)) = 64
- distance_4 = cost(left(3, 4), right(3, 1)) = 9

disparity = argmin(cost(left(3, 4), right(3, 4 – d))) = 0

Problem 01

❖ Step 3: Depth Calculation



0	0	1	1	3	0
0	0	1	0	2	1
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

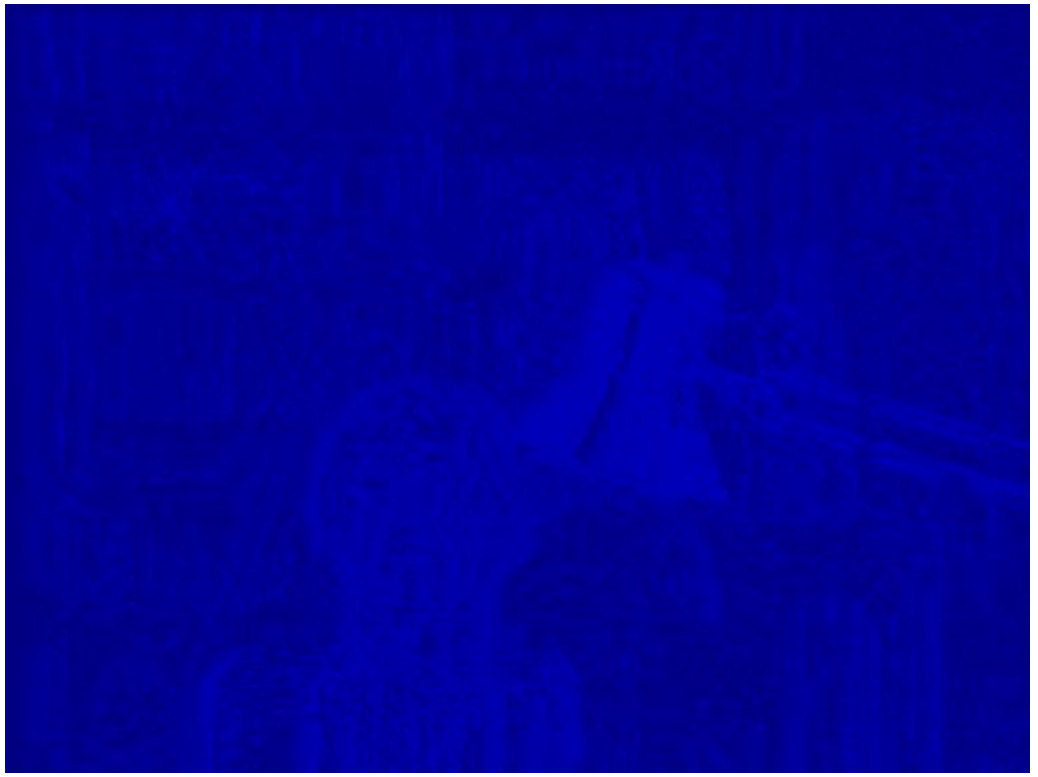
* $scale$

Disparity map (depth)

Problem 01

❖ Step 3: Why multiply by scale factor

scale serves for visualization purpose



Result images without scaling factor

Problem 01

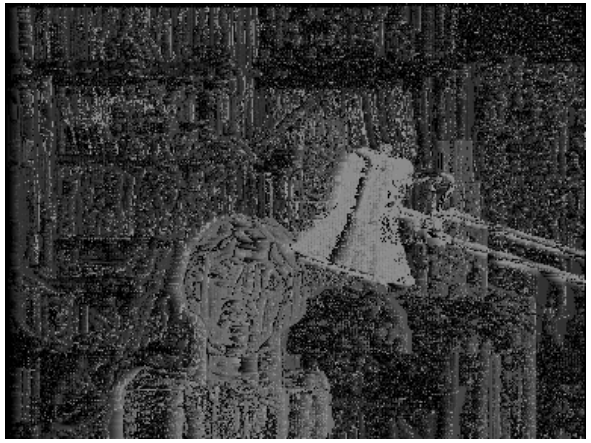
❖ Step 3: Why multiply by scale factor

scale serves for visualization purpose

scale = 1

scale = 3

scale = 6



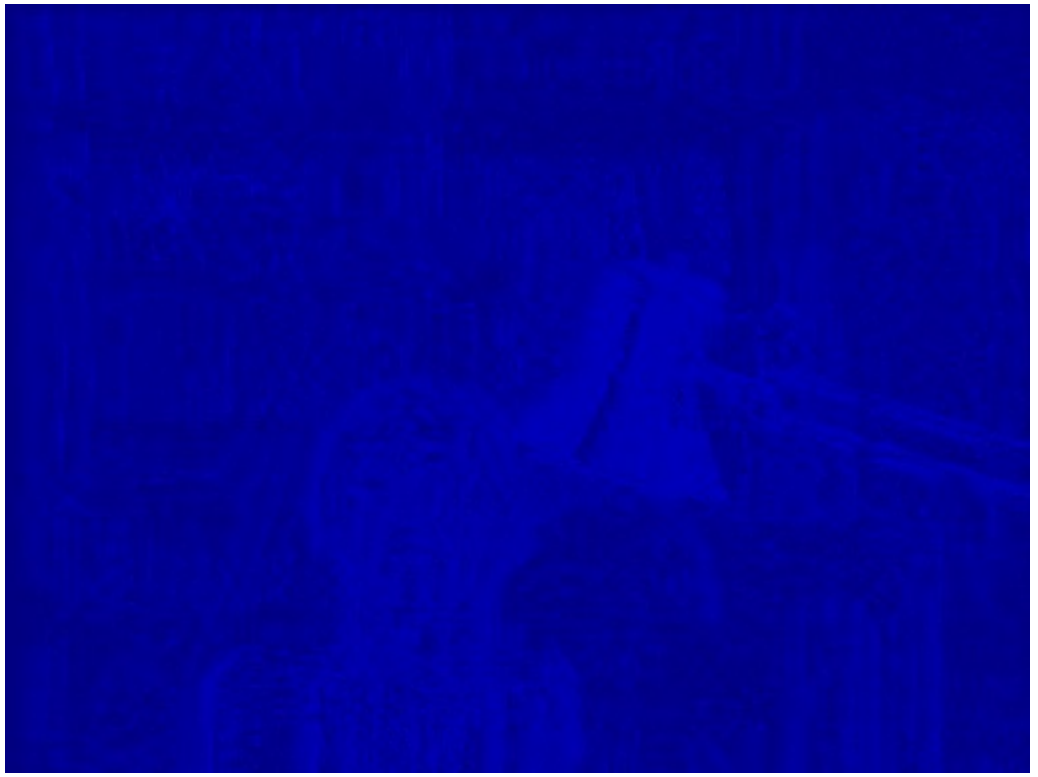
scale = 10

Proper scale value results in good looking image

Problem 01

❖ Step 3: Why multiply by scale factor

scale serves for visualization purpose

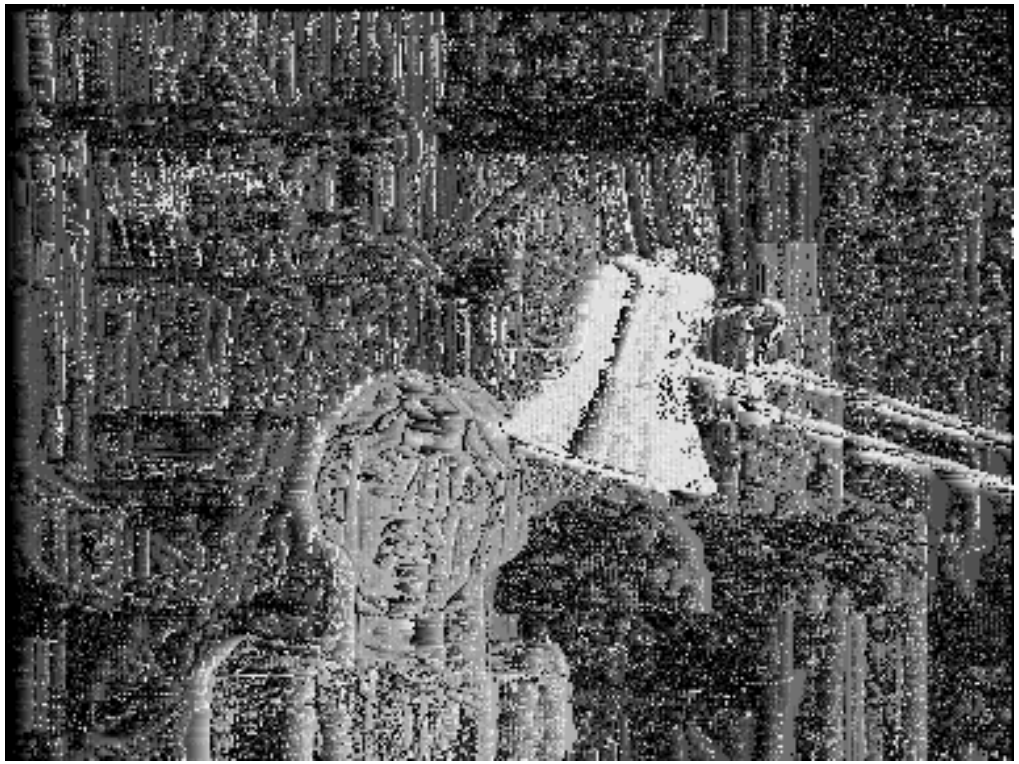


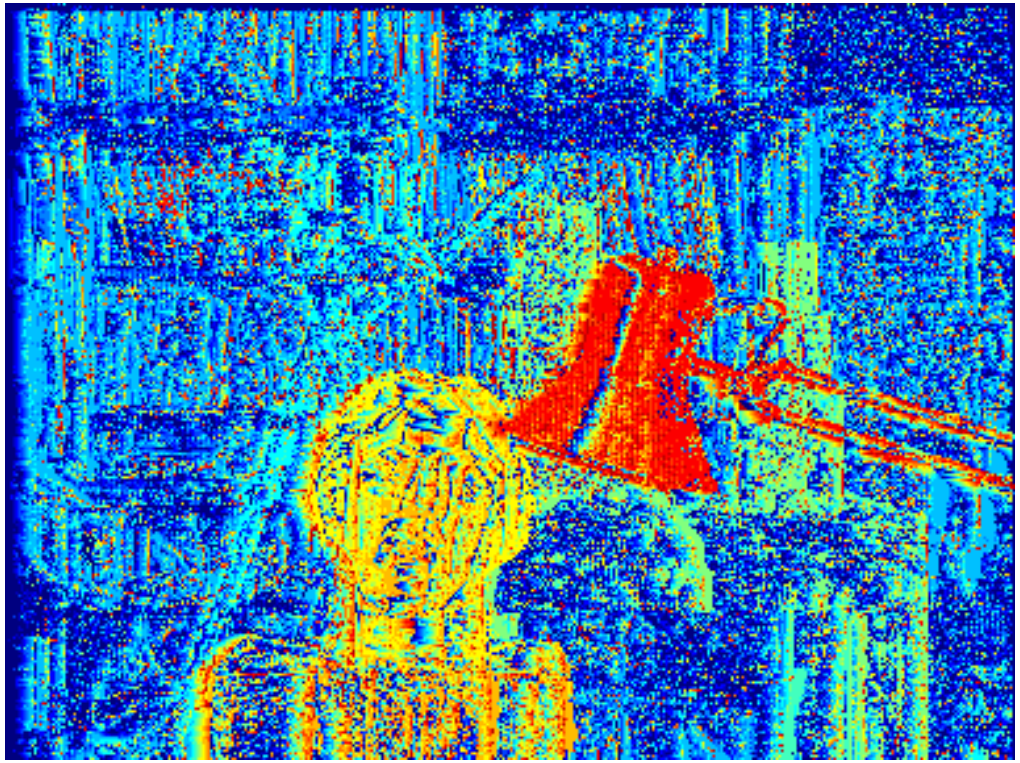
Result images without scaling factor

Problem 01

❖ Step 3: Why multiply by scale factor

scale serves for visualization purpose





Result images with a scaling factor of 16

Problem 01

❖ Code Implementation

❖ Define cost functions

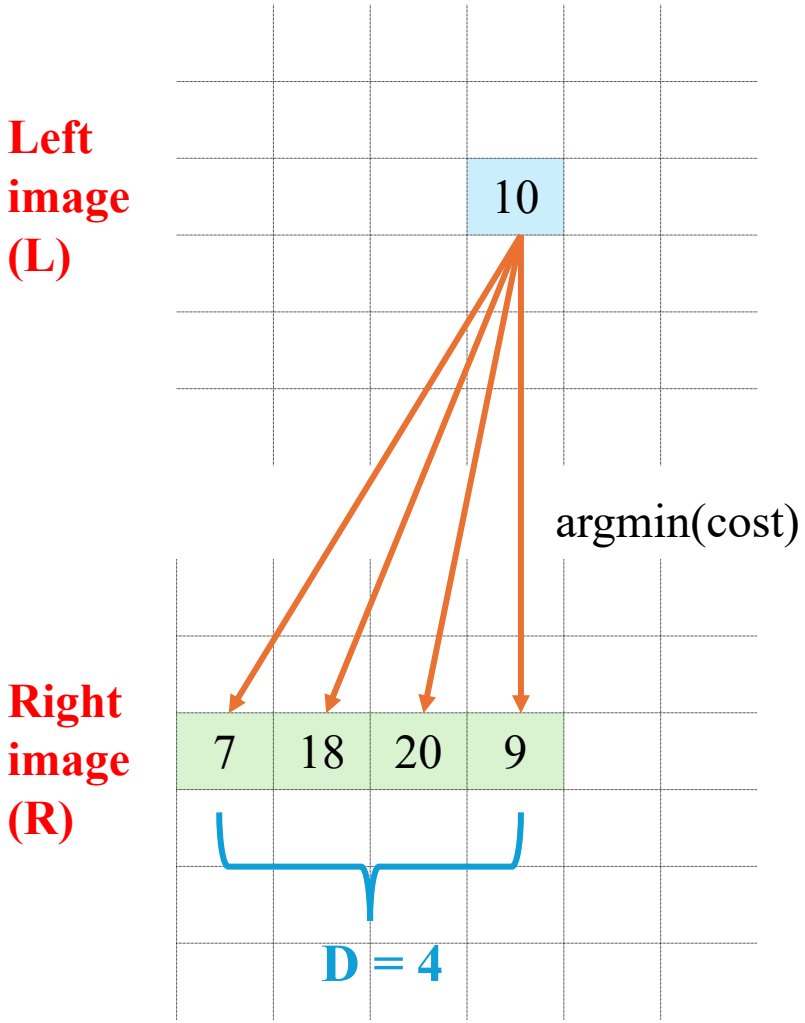
```python
import cv2
import numpy as np

def l1_distance(x, y):
    return abs(x - y)

def l2_distance(x, y):
    return (x - y) ** 2
```

❖ Read images and create zero matrix

```python
# 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 = 10
max_value = 255
```

Problem 01

❖ Code Implementation

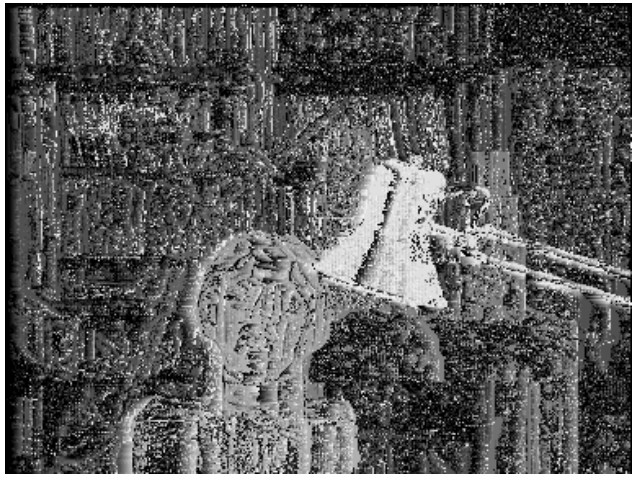


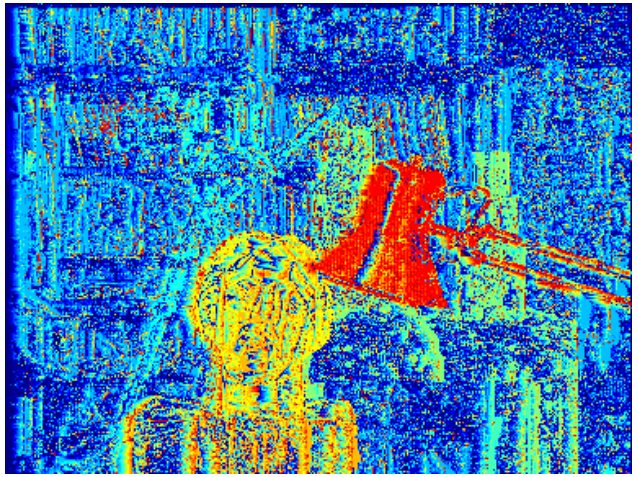

```python
for y in range(height):
    for x in range(width):
        # Find j where cost has minimum value
        disparity = 0
        cost_min  = max_value

        for j in range(disparity_range):
            cost = max_value if (x - j) < 0 \
                             else l1_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
```

Problem 01

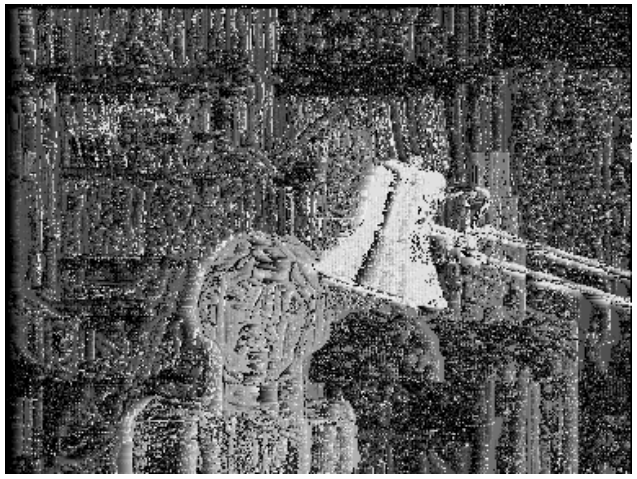
❖ Results

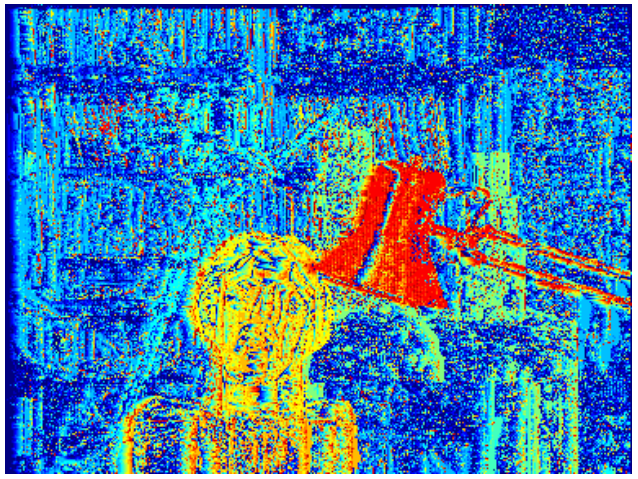
L1 Result:





L2 Result:





Problem 01

❖ Improvement ?

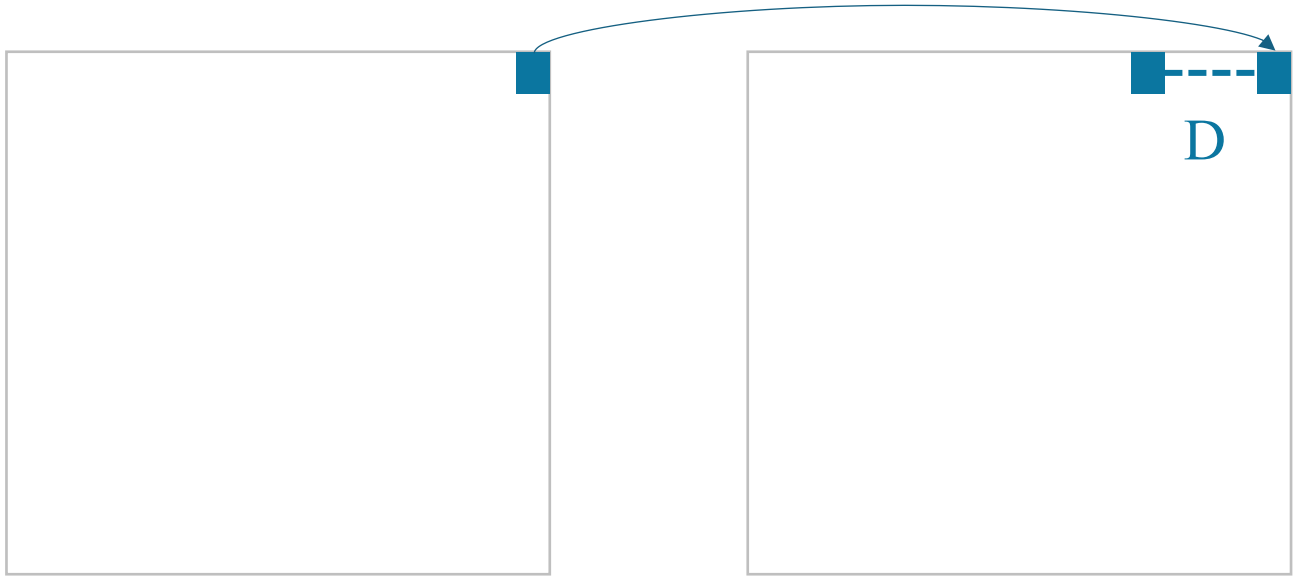
```
for y in range(height):
    for x in range(width):
        # Find j where cost has minimum value
        disparity = 0
        cost_min  = max_value

        for j in range(disparity_range):
            cost = max_value if (x - j) < 0 \
                             else l1_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
```

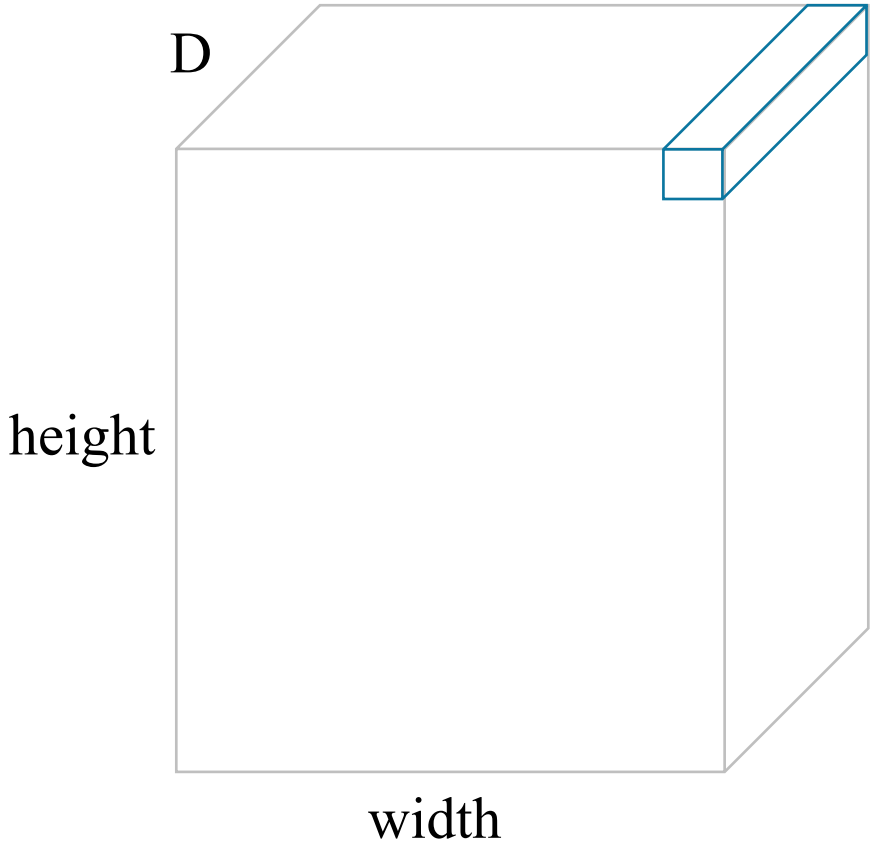
```
Saving result...
Done.
CPU times: user 3.54 s, sys: 51.8 ms, total: 3.59 s
Wall time: 3.71 s
```

Is there a way to reduce time execution?
⇒ Remove for-loop ?
⇒ Vectorization.

Problem 01

❖ Vectorization Approach





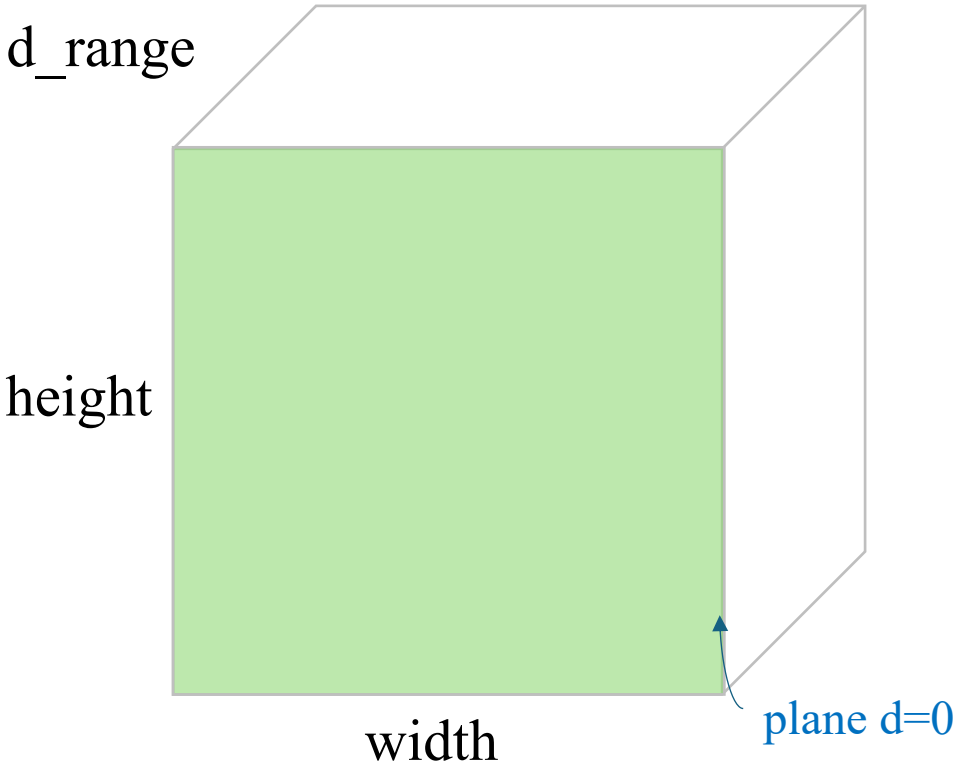
Normal approach

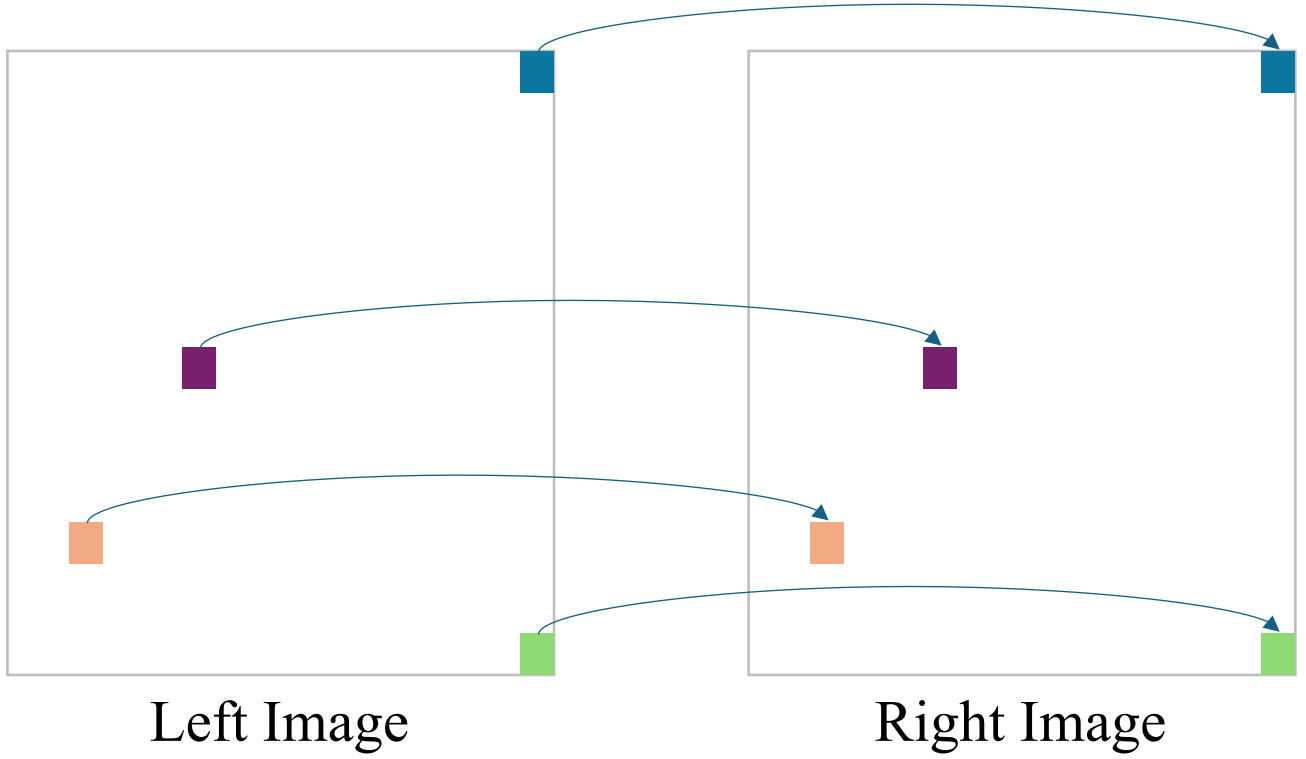
Each pixel p in the left image has D candidate pixel q in the right image.

Then, D cost values are computed from D pairs (**p**, **q**)

Problem 01

❖ Vectorization Approach





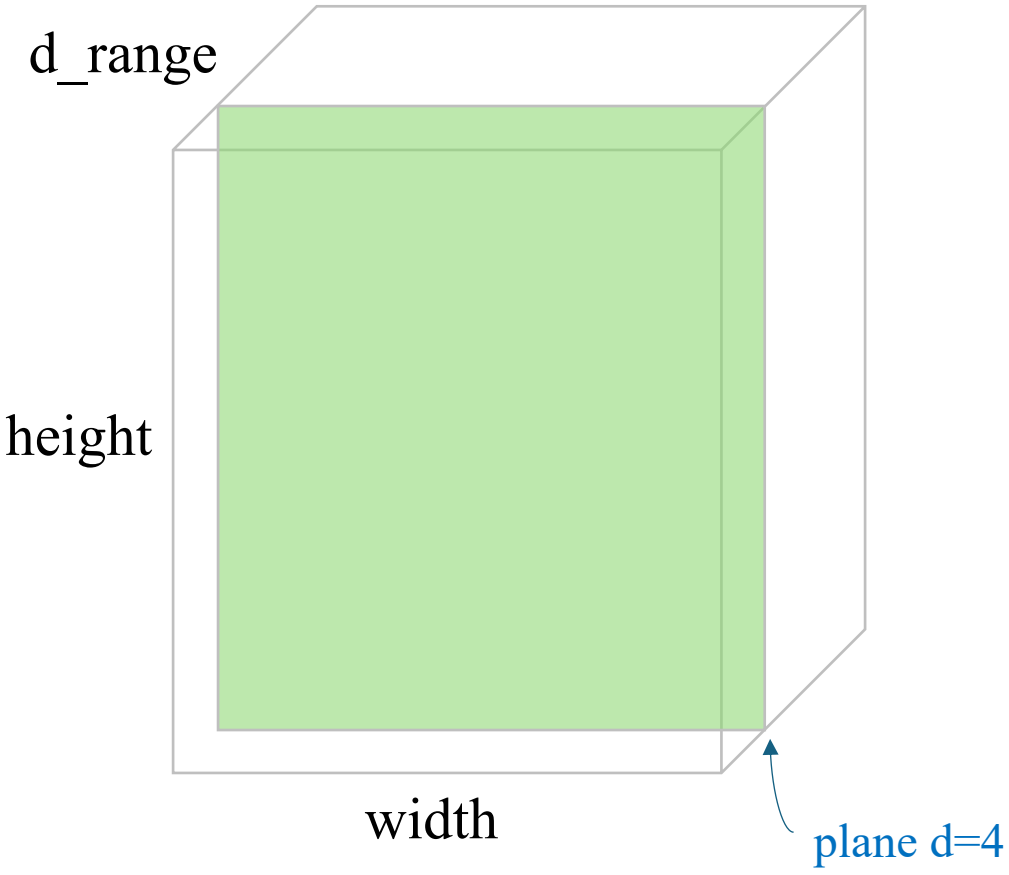
Given C_d is a cost plane for a disparity d

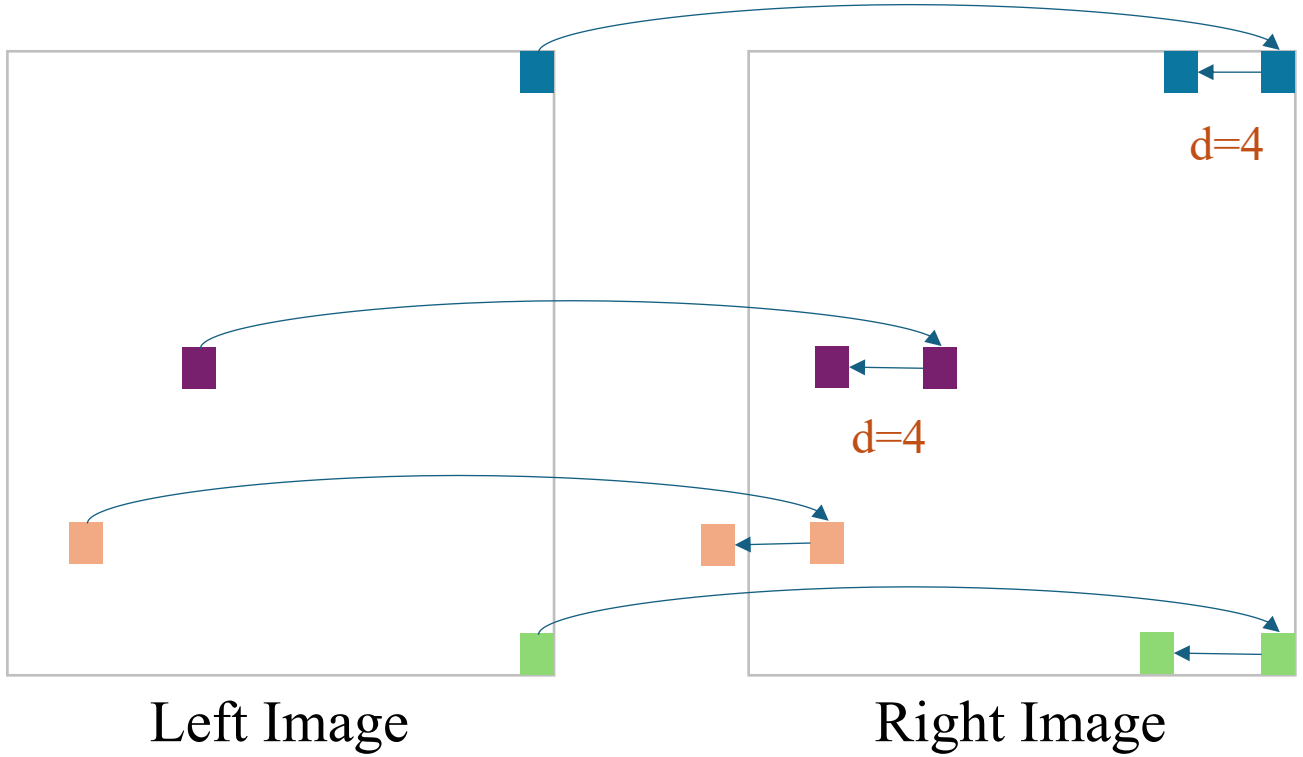
$C_0 = |L - R|$

Cost values for a disparity d=0

Problem 01

❖ Vectorization Approach

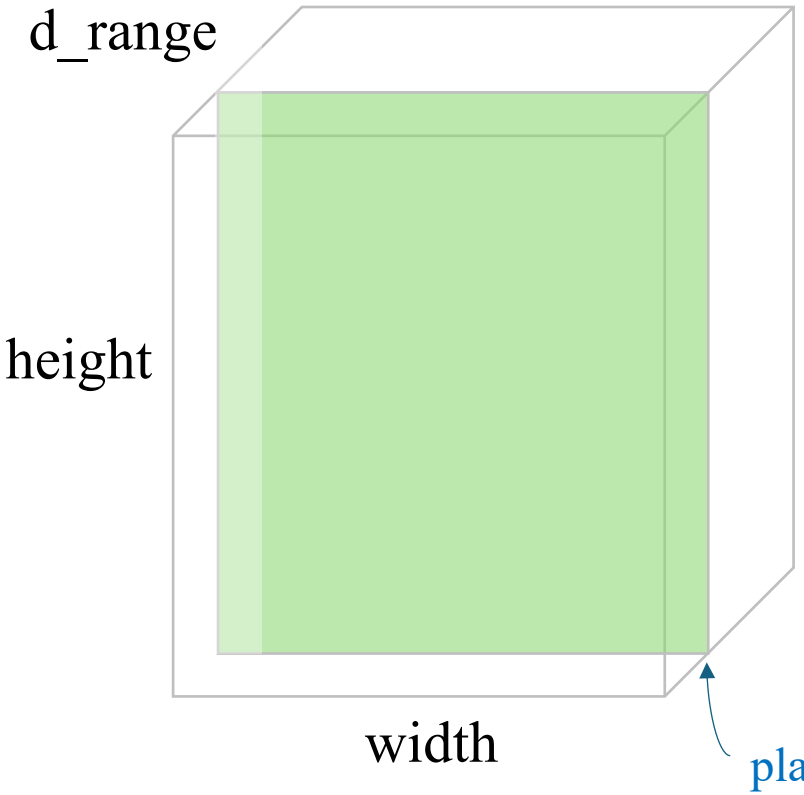


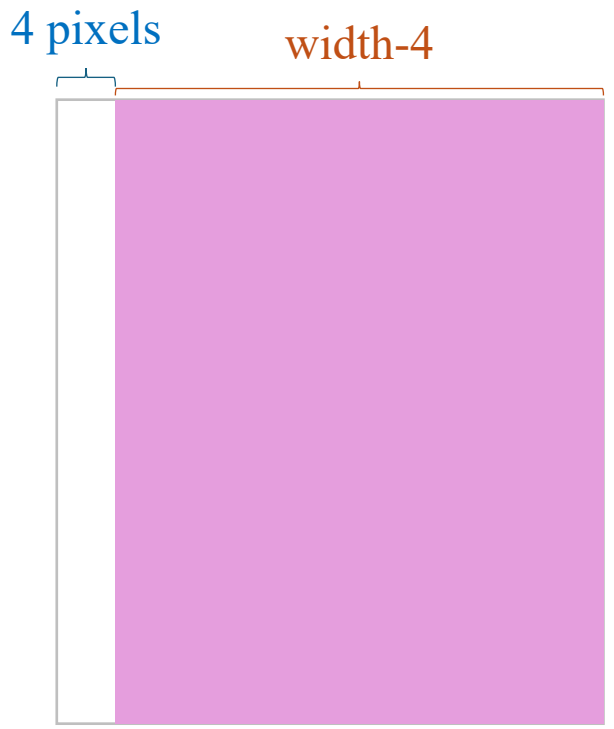


Cost values for a disparity d=4

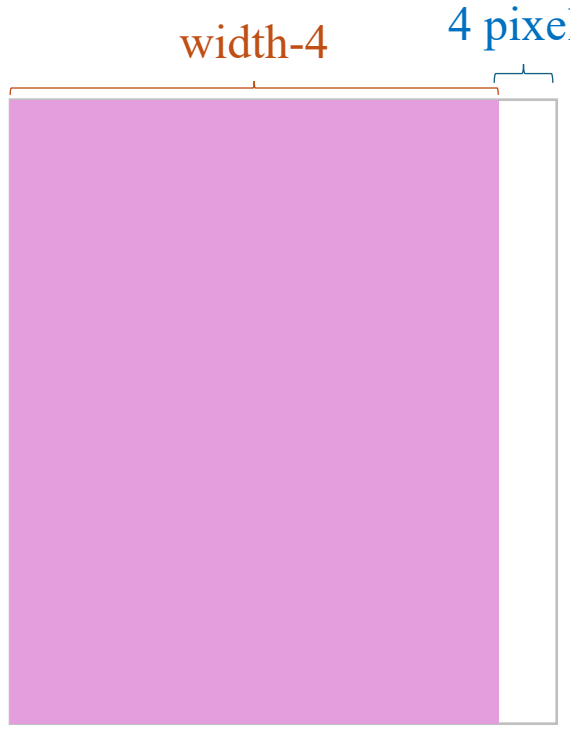
Problem 01

❖ Vectorization Approach





Left Image



Right Image

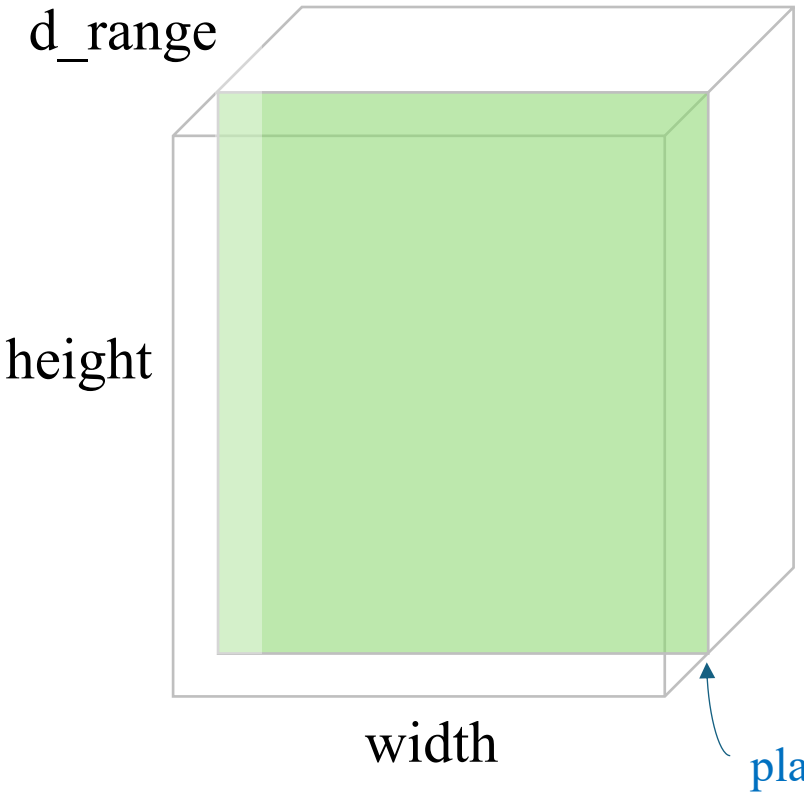
Given C_d is a cost plane for a disparity d

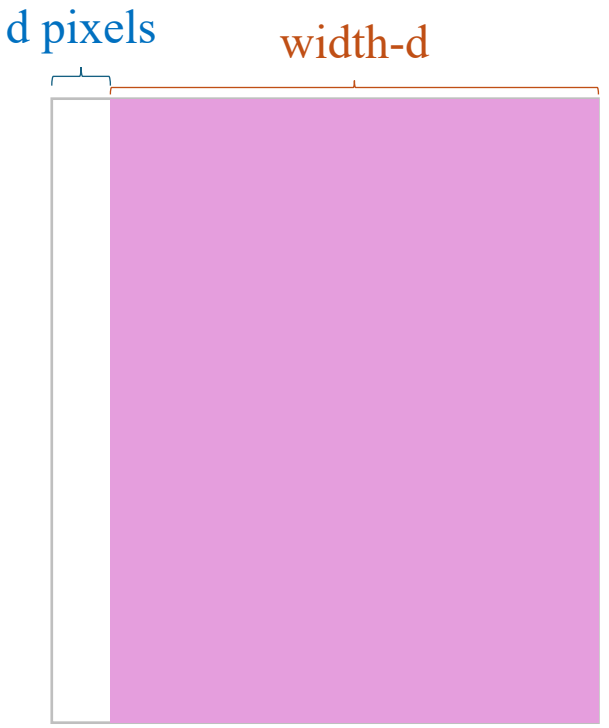
$$C_4 = |L[:, 4:W] - R[:, 0:W-4]|$$

Cost values for a disparity d=4

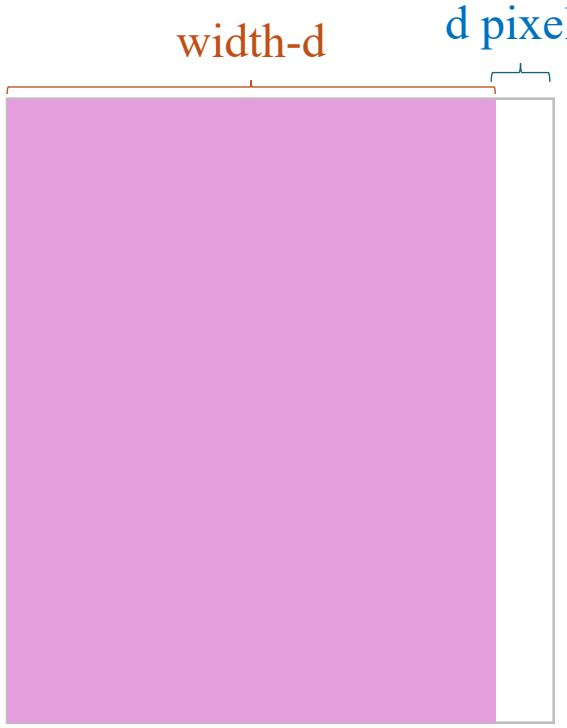
Problem 01

❖ Vectorization Approach





Left Image



Right Image

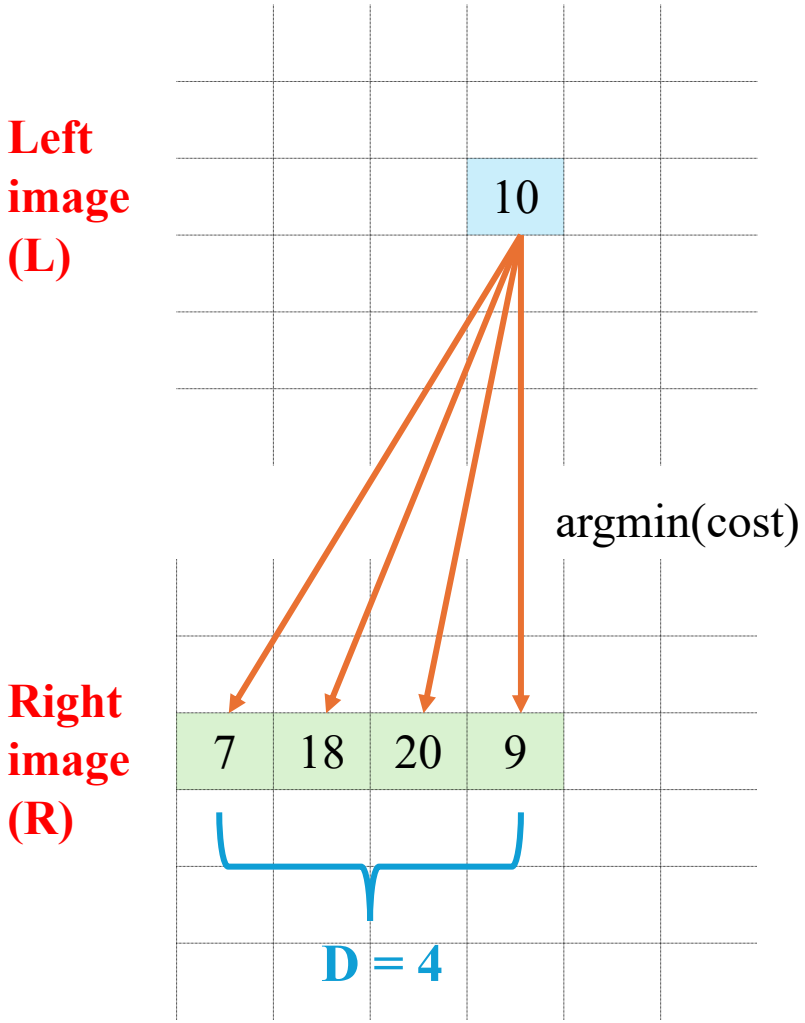
Given C_d is a cost plane for a disparity d

$$C_4 = |L[:, d:W] - R[:, 0:W-d]|$$

Cost values for a disparity d

Problem 01

❖ Code vectorization Implementation




```python
# Precompute the cost for all disparities
costs = np.full((height, width, disparity_range), max_value, dtype=np.float32)
for j in range(disparity_range):
    left_d = left[:,j:width]
    right_d = right[:,0:width-j]
    costs[:, j:width, j] = l1_distance(left_d, right_d)

# Find the disparity with the minimum cost
min_cost_indices = np.argmin(costs, axis=2)

# Set the disparity map
depth = min_cost_indices * scale
depth = depth.astype(np.uint8)
```

Problem 01

❖ Time comparison

```
%%time
# Nested for-loop approach
left_img_path = 'tsukuba/left.png'
right_img_path = 'tsukuba/right.png'
disparity_range = 16
pixel_wise_matching_l1(
    left_img_path,
    right_img_path,
    disparity_range,
    save_result=True
)
```

```
Saving result...
Done.
CPU times: user 3.54 s, sys: 51.8 ms, total: 3.59 s
Wall time: 3.71 s
```

```
%%time
# Vectorization approach
left_img_path = 'tsukuba/left.png'
right_img_path = 'tsukuba/right.png'
disparity_range = 16
pixel_wise_matching_l1(
    left_img_path,
    right_img_path,
    disparity_range,
    save_result=True
)
```

```
Saving result...
Done.
CPU times: user 42.2 ms, sys: 7.4 ms, total: 49.6 ms
Wall time: 97.4 ms
```

Problem 02

Problem 02

❖ Problem Statement

Build a function to calculate the disparity map of two input stereo images (left image (L) and right image (R)) by window-based matching method. The calculation steps in this method can be described through the following steps:

1. Read the left image (left) and the right image (right) as grayscale image and convert them to np.float32.
2. Initialize two variables height, width with values equal to the height and width of the left image.
3. Initialize a zero-zero matrix (depth) with shape equal to (height, width).
4. Calculate the half size of the window from the center to the edge of the window (of size $k \times k$) according to the formula $kernel\_half = \frac{(k-1)}{2}$ (round to integer).
5. For each pixel at position (h, w); ($h \in [kernel\_half, height - kernel\_half], w \in [kernel\_half, width - kernel\_half]$ traverse from left to right, top to bottom), perform the following steps. :

Problem 02

❖ Problem Statement

(a) Sum the costs (L1 or L2) between the pairs of pixels $left[h + v, w + u]$ and $right[h + v, w + u - d]$ (where $d \in [0, disparity\_range]$ (In this problem, $disparity\_range = 64$) and $u, v \in [-kernel\_half, kernel\_half]$) is in the window area with the center being the position of the pixel under consideration. If $(w + u - d) < 0$, assign the cost value = max_cost (max $_cost = 255$ if using L1 or max $_cost = 255^2$ if using L2).

(b) With the list of calculated costs, choose the d ($d_{optimal}$) value where the cost is the smallest.

(c) Assign $depth[h, w] = d_{optimal} \times scale$. (In this problem, $scale = 3$).

Problem 02

❖ Problem Statement

In this problem (as well as the rest), we run on **Aloe** stereo images. You can download from here.

Left Image

Right Image

Problem 02

❖ Method: Window-based Matching

L is the left image

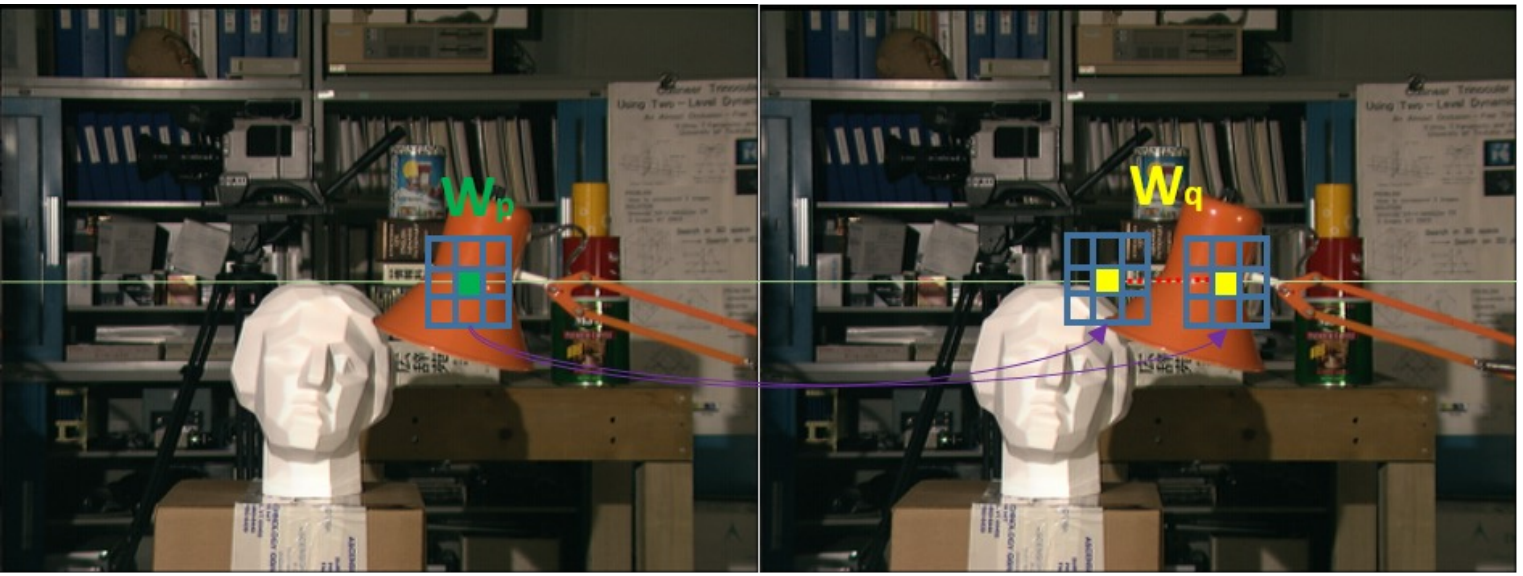
R is the right image

$L(\boldsymbol{p})$ is the (vector) value of **p**

$\boldsymbol{p} = \begin{bmatrix} x_p \\ y_p \end{bmatrix}$ $W_{\boldsymbol{p}}$ is a window centered at p

$\boldsymbol{d} = \begin{bmatrix} d \\ 0 \end{bmatrix}$ $C_2(\boldsymbol{p}, \boldsymbol{d}) = \sum_{\boldsymbol{u} \in W_{\boldsymbol{p}}} |L(\boldsymbol{u}) - R(\boldsymbol{u} - \boldsymbol{d})|$

$d \in D$



Left Image Right Image

Finding d so that $C_2(\boldsymbol{p}, \boldsymbol{d})$ is minimum.

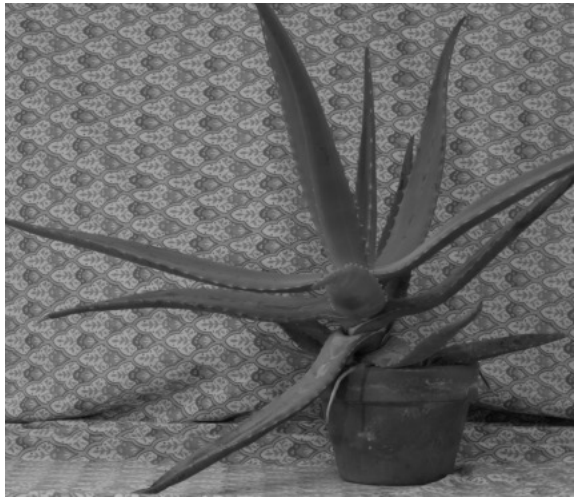
$$d = \underset{d \in D}{\operatorname{argmin}}\left(C_2(\boldsymbol{p}, \boldsymbol{d})\right)$$

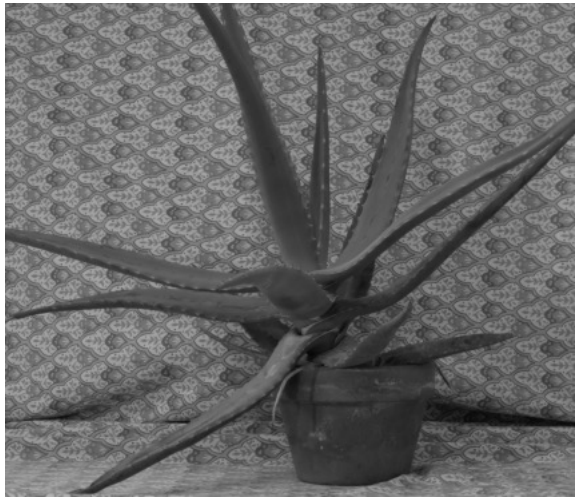
Then, d is the value for the pixel **p** in disparity map

Problem 02

❖ Step 1: Type Conversion

To grayscale



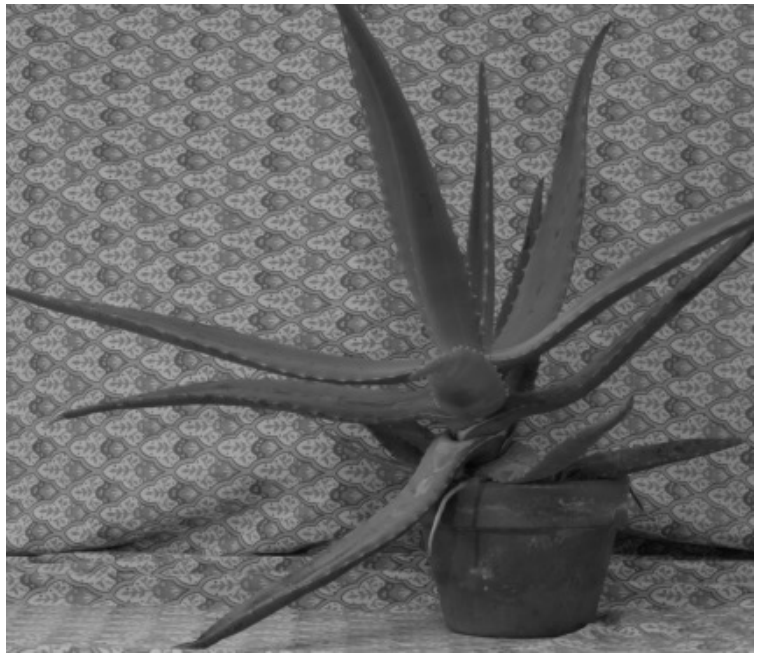


Type conversion

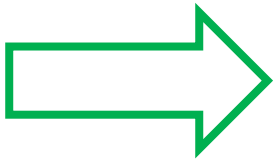
np.uint8 to np.float32

Problem 02

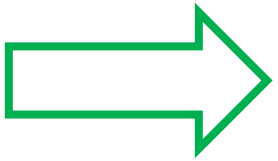
❖ Step 2: Zeros Matrix



Get height, width



height: 370
width: 427



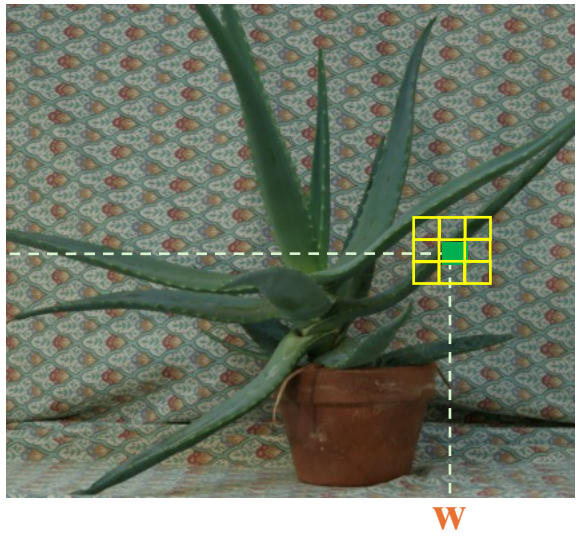
```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

Create zero matrix with height, width in np.uint8 (depth)

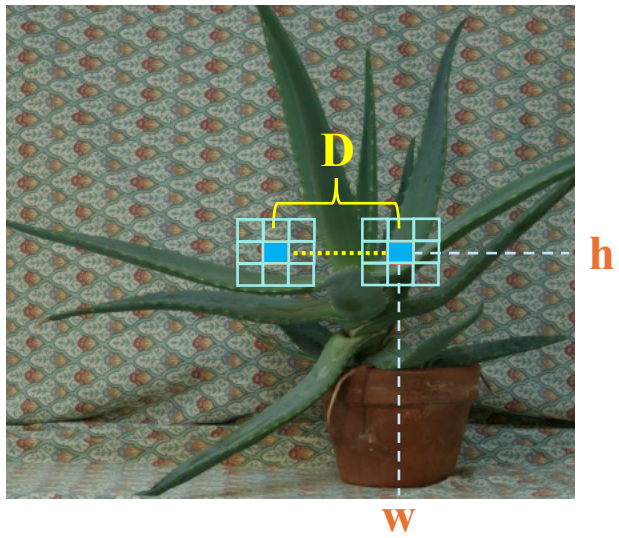
Problem 02

❖ Step 3: Depth Calculation

❖ **For each (h, w) point:**



Left image (L)



Right image (R)

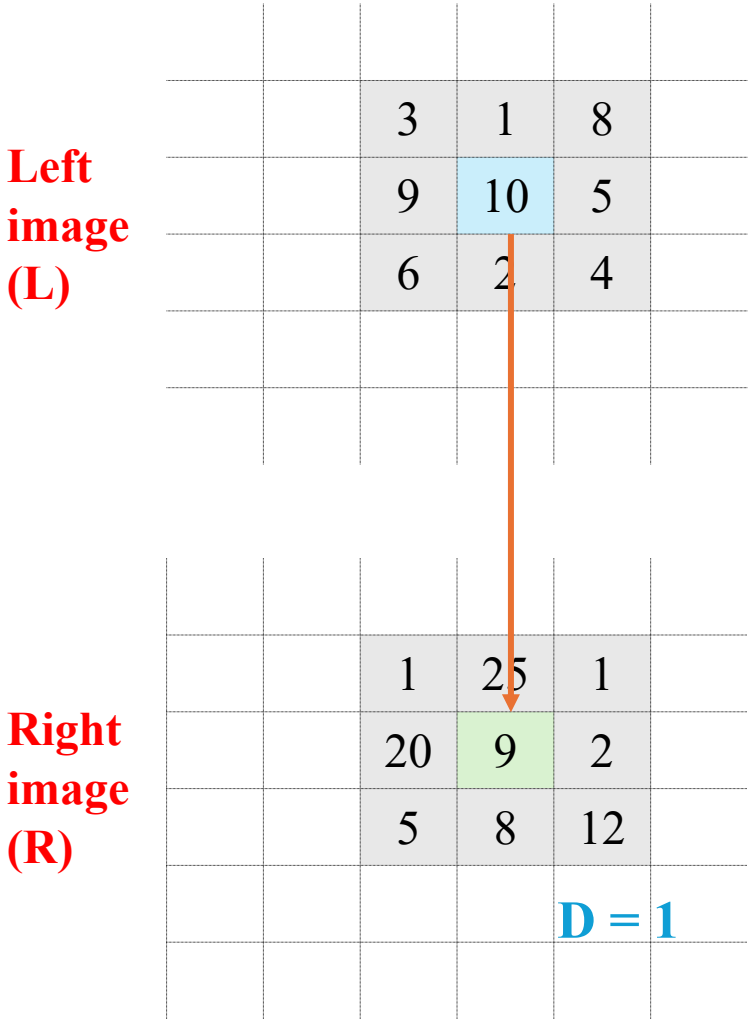
window(L(h, w)), window(R(h, w))

$$disparity = \underset{d}{argmin}\left(\sum_{u,v} cost(L[h+v, w+u], R[h+v, w+u-d])\right) \{d \in [0, D]; u, v \in \left[-\frac{k\_size-1}{2}, \frac{k\_size-1}{2}\right]\}$$

$$depth[h, w] = disparity \times scale$$

Problem 02

❖ Step 3: Depth Calculation



$$disparity = \underset{d}{argmin}\left(\sum_{u,v} cost(L[h+v, w+u], R[h+v, w+u - d])\right) \{d \in [0, D]; u, v \in \left[-\frac{k\_size-1}{2}, \frac{k\_size-1}{2}\right]\}$$

$$L1 = cost(x, y) = |x - y|$$

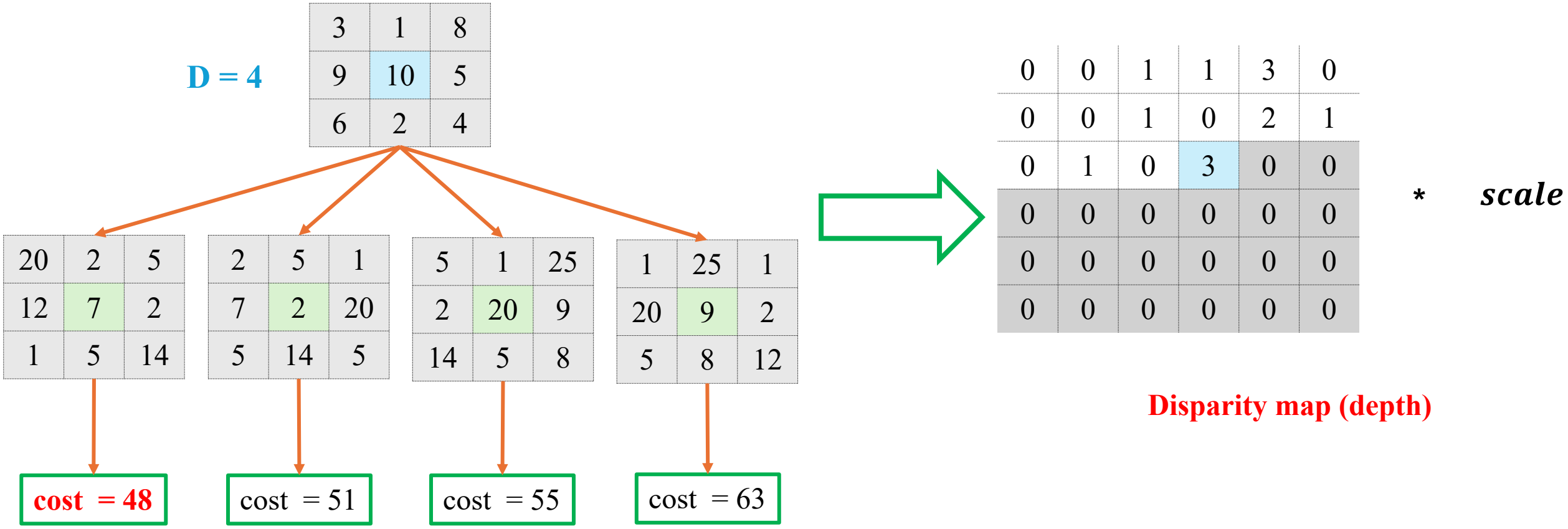
$$L2 = cost(x, y) = (x - y)^2$$

$$cost\left(\begin{array}{|c|c|c|}\hline 3 & 1 & 8 \\ \hline 9 & 10 & 5 \\ \hline 6 & 2 & 4 \\ \hline\end{array}, \begin{array}{|c|c|c|}\hline 1 & 25 & 1 \\ \hline 20 & 9 & 2 \\ \hline 5 & 8 & 12 \\ \hline\end{array}\right)$$

$$\sum_{u,v} cost(L[h+v, w+u], R[h+v, w+u])$$

Problem 02

❖ Step 3: Depth Calculation



Problem 02

❖ Code Implementation

❖ **Define cost functions**

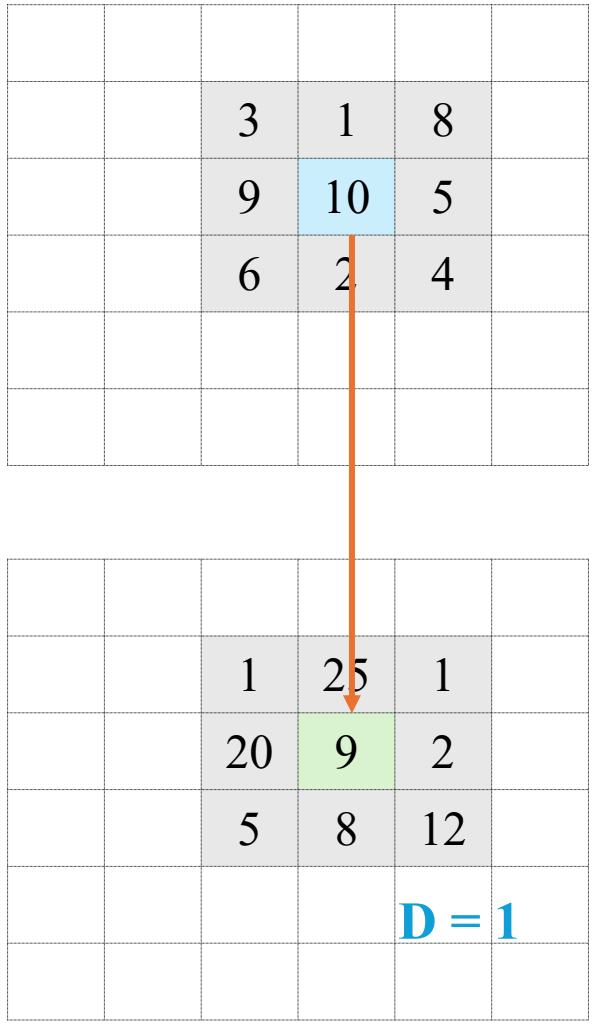
```python
import cv2
import numpy as np

def l1_distance(x, y):
    return abs(x - y)

def l2_distance(x, y):
    return (x - y) ** 2
```

❖ **Read image, create zero matrix & caculate half kernel size**

```python
# 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
```

Problem 02

❖ Code Implementation

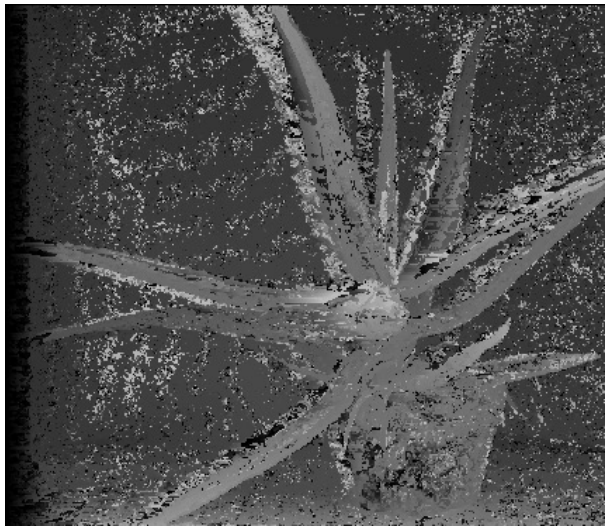


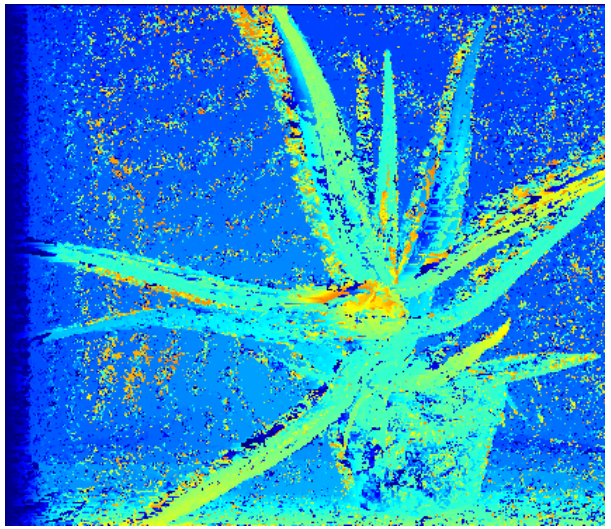

```python
for y in range(kernel_half, height-kernel_half):
    for x in range(kernel_half, width-kernel_half):

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

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

            for v in range(-kernel_half, kernel_half + 1):
                for u in range(-kernel_half, kernel_half + 1):
                    value = max_value
                    if (x + u - j) >= 0:
                        value = l2_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
```

Problem 02

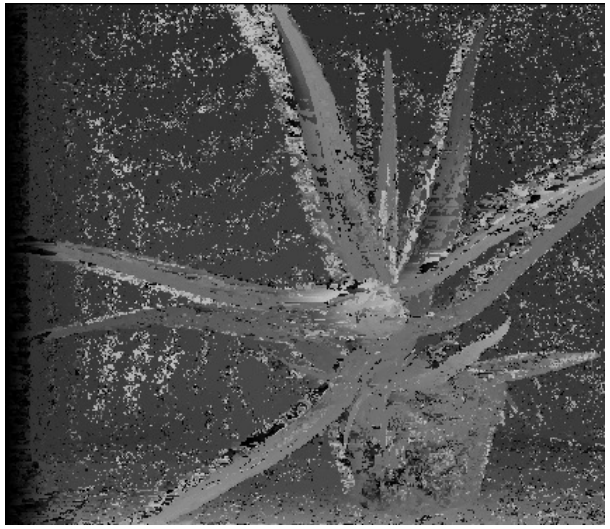
❖ Results

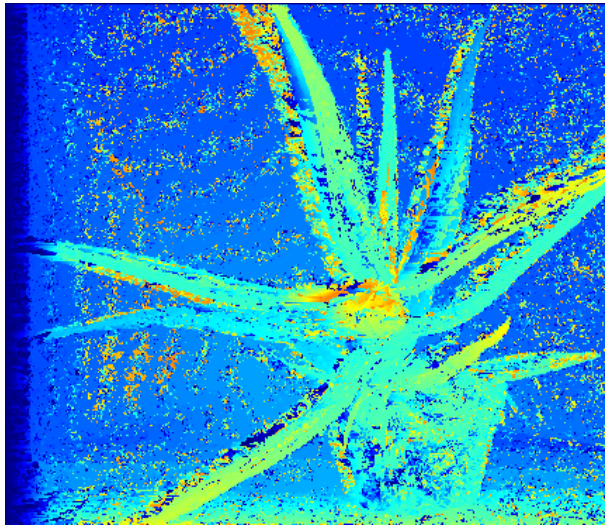
L1 Result:





L2 Result:





Problem 03

Problem 03

❖ Problem Statement

When using the disparity map calculation function built in **Problem 2** for the pair of images **Aloe_left_1.png** and **Aloe_right_2.png** with input parameters $disparity\_range = 64$ and $kernel\_size = 5$ in both L1 and L2 cost, we get a disparity map result of part is noisy.

It can be seen that with the change of input stereo images values, the disparity map results have somewhat deteriorated (with noise). Please use the code in Problem 2 to generate a disparity map with this setting and explain (using markdown) why this is happening?

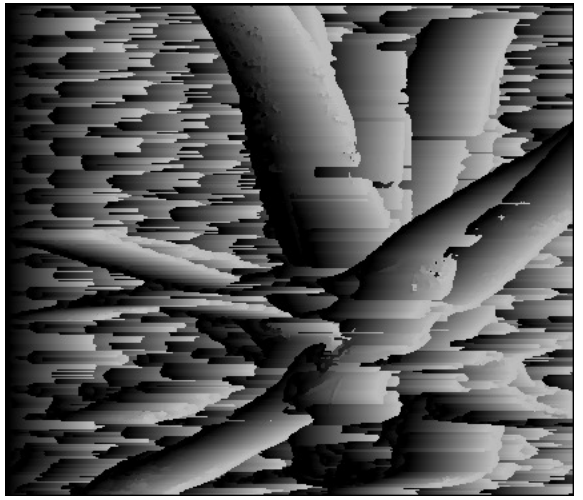
Problem 03

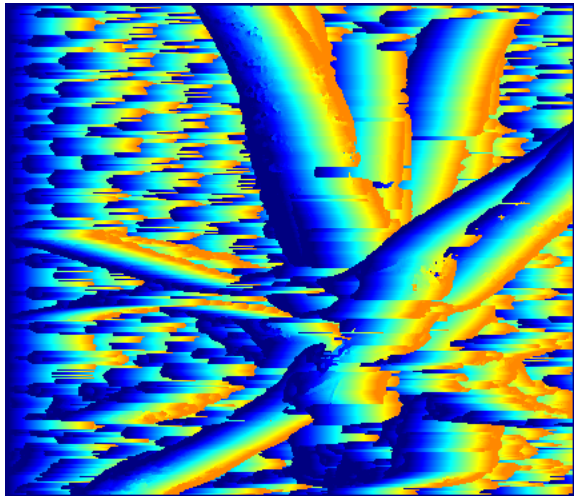
❖ Problem

Left image (L)

Right image (R)

Window-based matching





Problem 03

❖ Solution

An Image (L)

L + 10

L + 50

L + 100

Problem 03

❖ Solution

Let:
- Window A = x = [1, 2, 3, 4]
- Window B1 = y = [1, 1, 2, 5]
- Window B2 = z = [0, 0, 1, 0]

Compute L1 cost:
(A, B1) = $\sum_i |x_i - y_i| = 0 + 1 + 1 + 1 = 3$
(A, B2) = $\sum_i |x_i - y_i| = 1 + 2 + 2 + 4 = 9$

Let:
- Window A = x = [1, 2, 3, 4]
- Window B1 = y = [1, 1, 2, 5] + 50 = [51, 51, 52, 55]
- Window B2 = z = [0, 0, 1, 0] + 50 = [50, 50, 51, 50]

Compute L1 cost:
(A, B1) = $\sum_i |x_i - y_i| = 50 + 49 + 49 + 51 = 199$
(A, B2) = $\sum_i |x_i - y_i| = 49 + 48 + 48 + 46 = 191$

➔ The result will be changed badly as other side image linearly changes.

➔ L1 and L2 don't have the **'invariant to linear changes'** property.

QUIZ

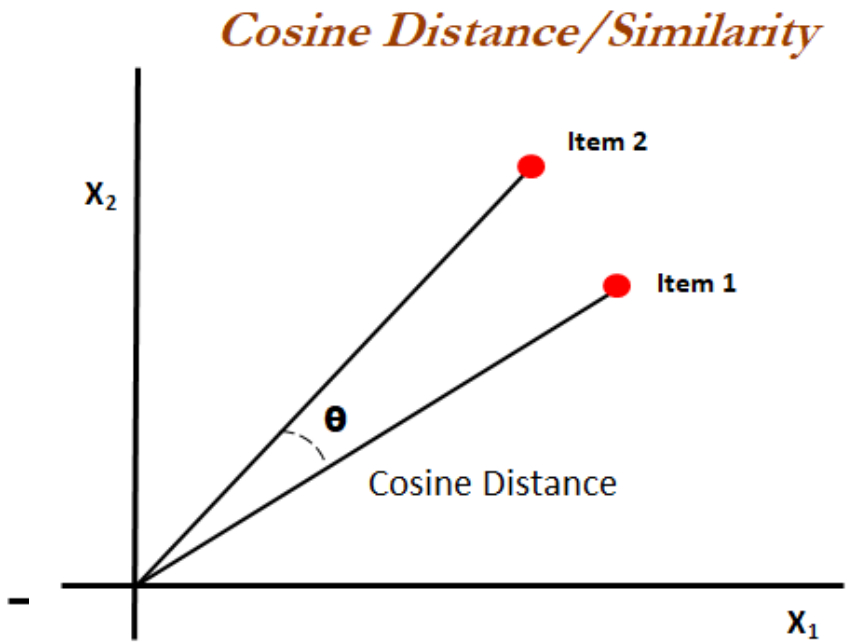
Problem 04

Problem 04

❖ Problem Statement

Based on the window-based matching disparity map function in **Problem 2** and treat windows as vectors, implement **Cosine Similarity** in calculating the correlation between left and right image pixels to solve the problem in **Problem 3**. The formula for cosine similarity are described as follows:

$$cosine\_similarity(x, y) = \frac{x \cdot y}{\|x\|\|y\|} = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}}$$



https://www.oreilly.com/api/v2/epubs/9781788295758/files/assets/2b4a7a82-ad4c-4b2a-b808-e423a334de6f.png

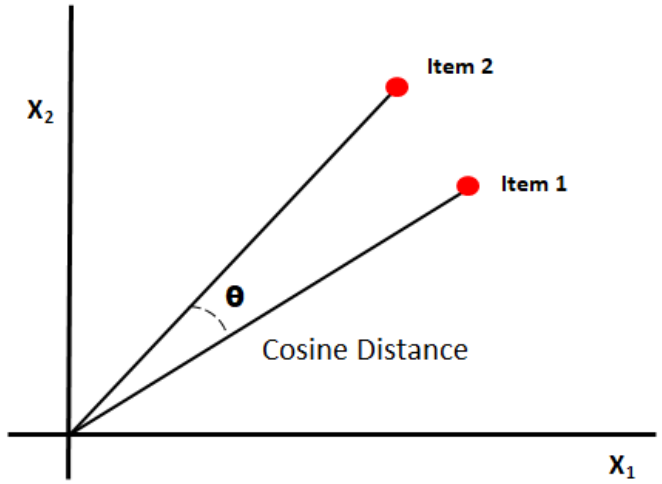
Problem 04

❖ Method: Idea

$$C_2(\boldsymbol{p}, \boldsymbol{d}) = \sum_{\boldsymbol{u} \in W_{\boldsymbol{p}}} |L(\boldsymbol{u}) - R(\boldsymbol{u} - \boldsymbol{d})|$$

$$cosine\_similarity(x, y) = \frac{x \cdot y}{\|x\|\|y\|} = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}}$$

Cosine Distance/Similarity



Left Image Window

3	1	8
9	10	5
6	2	4

cost()

Right Image Windows

2	5	1		1	25	1
7	2	20	• • •	20	9	2
5	14	5		5	8	12

https://www.oreilly.com/api/v2/epubs/9781788295758/files/assets/2b4a7a82-ad4c-4b2a-b808-e423a334de6f.png