1.

import cv2

import numpy as np

# load image

input\_image = cv2.imread('/Users/kasi/Downloads/nadia2.jpg', cv2.IMREAD\_GRAYSCALE)

template = cv2.imread('/Users/kasi/Downloads/nadiatemplate.jpg', cv2.IMREAD\_GRAYSCALE)

# search matches

result = cv2.matchTemplate(input\_image, template, cv2.TM\_CCOEFF\_NORMED)

# normalize result

normalized\_result = cv2.normalize(result, None, 0, 255, cv2.NORM\_MINMAX, dtype=cv2.CV\_8U)

# find min max

min\_val, max\_val, min\_loc, max\_loc = cv2.minMaxLoc(result)

# white retangle over the image

top\_left = max\_loc

bottom\_right = (top\_left[0] + template.shape[1], top\_left[1] + template.shape[0])

cv2.rectangle(input\_image, top\_left, bottom\_right, 255, 2)

cv2.imshow('Detected', input\_image)

cv2.imshow('Error Map', normalized\_result)

cv2.waitKey(0)

cv2.destroyAllWindows()

A black and white striped object

Description automatically generated

2.

import cv2

import numpy as np

# load image

input\_image = cv2.imread('/Users/kasi/Downloads/nadia2.jpg', cv2.IMREAD\_GRAYSCALE)

template = cv2.imread('/Users/kasi/Downloads/nadiatemplate.jpg', cv2.IMREAD\_GRAYSCALE)

# search matches

result = cv2.matchTemplate(input\_image, template, cv2.TM\_CCOEFF\_NORMED)

# normalize result

normalized\_result = cv2.normalize(result, None, 0, 255, cv2.NORM\_MINMAX, dtype=cv2.CV\_8U)

# find min max

min\_val, max\_val, min\_loc, max\_loc = cv2.minMaxLoc(result)

# white retangle over the image

top\_left = max\_loc

bottom\_right = (top\_left[0] + template.shape[1], top\_left[1] + template.shape[0])

cv2.rectangle(input\_image, top\_left, bottom\_right, 255, 2)

cv2.imshow('Detected', input\_image)

cv2.imshow('Error Map', normalized\_result)

cv2.waitKey(0)

cv2.destroyAllWindows()

A green and black background with a square and a black square

Description automatically generated

3.

import cv2

import numpy as np

def ebma(target, anchor, block\_size, search):

motion\_vectors = []

for y in range(0, target.shape[0], block\_size):

for x in range(0, target.shape[1], block\_size):

best\_match = (0, 0)

min\_error = float('inf')

x\_start = max(x - search, 0)

x\_end = min(x + block\_size + search, anchor.shape[1])

y\_start = max(y - search, 0)

y\_end = min(y + block\_size + search, anchor.shape[0])

for yy in range(y\_start, y\_end - block\_size + 1):

for xx in range(x\_start, x\_end - block\_size + 1):

block\_target = target[y:y+block\_size, x:x+block\_size]

block\_anchor = anchor[yy:yy+block\_size, xx:xx+block\_size]

if block\_target.shape == block\_anchor.shape:

mse = np.mean((block\_target - block\_anchor) \*\* 2)

if mse < min\_error:

min\_error = mse

best\_match = (xx, yy)

motion\_vectors.append(((x, y), (best\_match[0] - x, best\_match[1] - y)))

return motion\_vectors

# load the video

cap = cv2.VideoCapture('/Users/kasi/Downloads/vb.mov')

block\_size = 8

search = 7

anchor\_frame = None

frame\_count = 0

while cap.isOpened():

ret, frame = cap.read()

if not ret:

break

target\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

if anchor\_frame is not None:

motion\_vectors = ebma(target\_frame, anchor\_frame, block\_size, search)

quiver = np.zeros((target\_frame.shape[0], target\_frame.shape[1], 3), dtype=np.uint8)

for (x, y), (dx, dy) in motion\_vectors:

cv2.arrowedLine(quiver, (x, y), (x + dx, y + dy), (0, 255, 0), 1, tipLength=0.3)

cv2.imshow('Anchor Frame', anchor\_frame)

cv2.imshow('Target Frame', target\_frame)

cv2.imshow('Motion Vectors', quiver)

cv2.waitKey(100)

frame\_count += 1

if frame\_count > 10:

break

anchor\_frame = target\_frame.copy()

cap.release()

cv2.destroyAllWindows()

A green and black pattern

Description automatically generatedA green and black background with arrows

Description automatically generated