Video Image Digital Processing

Assignment 3

Fotopoulos Ioannis AM: 387

Python Implementation

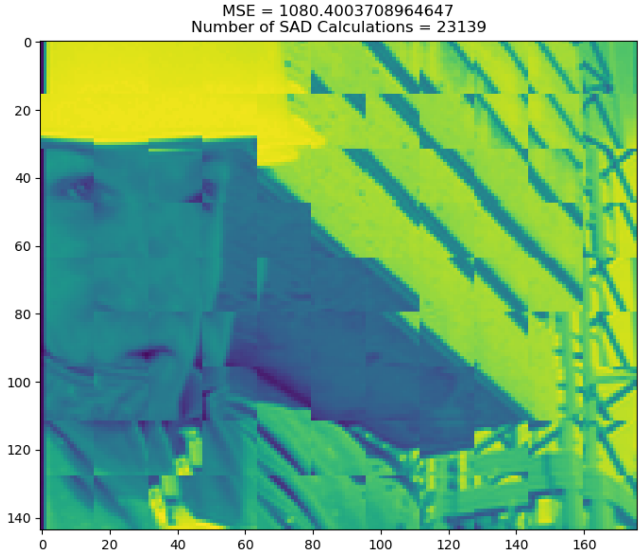
```
import cv2 as cv

import numpy as np

from matplotlib import pyplot as plt

import copy

frame178 = cv.imread('frame178.tif', 0)

frame179 = cv.imread('frame179.tif', 0)


SAD_Table = {}

Optimal_Directions = {}


n_rows, n_cols = frame178.shape

block_size = 16

range_search = 8

y_dir=x_dir=list(range(-range_search,range_search+1,1))


def SAD(frame1,frame2):

    return np.sum(np.absolute(frame1-frame2))


def MSE(estimation,frame):

    return np.sum((estimation-frame)**2)/(n_cols*n_rows)


#Exhaustive Search Routine

def EBMA(start_x,end_x,start_y, end_y,frame_prev,frame_cur):

    temp = frame_cur[start_x:end_x,start_y:end_y]
    cur_SAD = min_SAD = SAD(frame_prev[start_x:end_x,start_y:end_y],temp)
    opt_dir = ((start_x,start_y),(0,0))
    SAD_Table[(start_x,end_x,start_y, end_y,start_x,end_x,start_y, end_y)]=min_SAD
    #Exhaustive Search in any direction
    for x in x_dir:
        i = start_x + x
        ii = i + block_size
        if 0<=  i < n_rows and 0<= ii < n_rows:
            for y in y_dir:
                j = start_y + y
                jj = j + block_size
                if 0<= j < n_cols and 0<= jj < n_cols:
                    key = (start_x,end_x,start_y, end_y,i,ii,j,jj)
                    if key in SAD_Table:
                        cur_SAD = SAD_Table[key]
                    else:
                        cur_SAD = SAD(frame_prev[i:ii,j:jj],temp)
                        SAD_Table[key] = cur_SAD

                    if cur_SAD < min_SAD:
                        opt_dir = ((start_x,start_y),(x,y))
                        min_SAD = cur_SAD
    return opt_dir
#2D Log Search Routine
def two_D_log_search(start_x,end_x,start_y, end_y,frame_prev,frame_cur,step):
    center_start_x = start_x
    center_end_x = end_x
    center_start_y = start_y
    center_end_y = end_y
    temp = frame_cur[start_x:end_x,start_y:end_y]
    cur_SAD = min_SAD = 0
    opt_dir = ((start_x,start_y),(0,0))
    i = 0
    best_point = (start_x,start_y)
    while(True):
        #Initialize the cur SAD Error
        if i ==0:
            cur_SAD = min_SAD = SAD(frame_prev[center_start_x:center_end_x,center_start_y:center_end_y],temp)
            SAD_Table[(start_x,end_x,start_y, end_y,start_x,end_x,start_y, end_y)]=min_SAD
        #Search in the diamond point space
        for x in [-step,0,step]:
            i = center_start_x + x
            ii = i + block_size

            if np.fabs(i-start_x) <= range_search and np.fabs(ii-end_x) <= range_search and 0<=  i < n_rows and 0<= ii < n_rows:
                key = (start_x,end_x,start_y,end_y,i,ii,center_start_y,center_end_y)
                if key in SAD_Table:
                    cur_SAD = SAD_Table[key]
                else:
                    cur_SAD = SAD(frame_prev[i:ii,center_start_y:center_end_y],temp)
                    SAD_Table[key] = cur_SAD
                if cur_SAD < min_SAD:
                    min_SAD = cur_SAD
                    best_point = (i,center_start_y)
        for y in [-step,step]:
            j = center_start_y + y
            jj = j + +block_size
            if np.fabs(j-start_y) <= range_search and np.fabs(jj-end_y) <= range_search and 0<=  j < n_cols and 0<= jj < n_cols:
```

```
            key = (start_x,end_x,start_y, end_y,center_start_x,center_end_x,j,jj)
            if key in SAD_Table:
                cur_SAD = SAD_Table[key]
            else:

                cur_SAD = SAD(frame_prev[center_start_x:center_end_x,j:jj],temp)
                SAD_Table[key] = cur_SAD
            if cur_SAD < min_SAD:
                min_SAD = cur_SAD
                best_point = (center_start_x,j)
    #Termination step: Search the optimal block in 9 points
    if step <= 1:
        for x in [-step,step]:
            i = center_start_x + x
            ii = i + block_size
            if np.fabs(i-start_x) <= range_search and np.fabs(ii-end_x) <= range_search and 0<=  i < n_rows and 0<= ii < n_rows:
                for y in [-step,step]:
                    j = center_start_y + y
                    jj = j + +block_size
                    if np.fabs(j-start_y) <= range_search and np.fabs(jj-end_y) <= range_search and 0<=  j < n_cols and 0<= jj < n_cols:
                        key = (start_x,end_x,start_y,end_y,i,ii,j,jj)
                        if key in SAD_Table:
                            cur_SAD = SAD_Table[key]
                        else:
                            cur_SAD = SAD(frame_prev[i:ii,j:jj],temp)
                            SAD_Table[key] = cur_SAD
                        if cur_SAD < min_SAD:
                            min_SAD = cur_SAD
                            best_point = (i,j)
            break
        #Reduce the stepsize when the current best point has been moved to borders
        if np.fabs(start_x - best_point[0]) == range_search or np.fabs(start_y - best_point[1]) == range_search:
            step = step//2
        #and when the current best point has not been changed
        elif best_point[0] == center_start_x  and best_point[1] == center_start_y:
            step = step//2

        #update the center point
        center_start_x,center_start_y=best_point
        center_end_x = center_start_x + block_size
        center_end_y = center_start_y + block_size
        i += 1
    return ((start_x,start_y),(best_point[0]-start_x,best_point[1]-start_y))

#Three Step Routine
def three_step_search(start_x,end_x,start_y, end_y,frame_prev,frame_cur,step):
    center_start_x = start_x
    center_end_x = end_x
    center_start_y = start_y
    center_end_y = end_y
    temp = frame_cur[start_x:end_x,start_y:end_y]
    cur_SAD = min_SAD = 0
    opt_dir = ((start_x,start_y),(0,0))
    i = 0
    best_point = (start_x,start_y)
    while(True):
        #Initialization Step
        if i ==0:
```

```
    cur_SAD = min_SAD = SAD(frame_prev[center_start_x:center_end_x,center_start_y:center_end_y],temp)
    SAD_Table[(start_x,end_x,start_y, end_y,start_x,end_x,start_y, end_y)]=min_SAD
#Search the optimal block in the 9 point space
for x in [-step,0,step]:
    i = center_start_x + x
    ii = i + block_size
    if np.fabs(i-start_x) <= range_search and np.fabs(ii-end_x) <= range_search and 0<=  i < n_rows and 0<= ii < n_rows:
        for y in [-step,0,step]:
            j = center_start_y + y
            jj = j + +block_size
            if np.fabs(j-start_y) <= range_search and np.fabs(jj-end_y) <= range_search and 0<=  j < n_cols and 0<= jj < n_cols:
                key = (start_x,end_x,start_y,end_y,i,ii,j,jj)
                if key in SAD_Table:
                    cur_SAD = SAD_Table[key]
                else:
                    cur_SAD = SAD(frame_prev[i:ii,j:jj],temp)
                    SAD_Table[key] = cur_SAD
                if cur_SAD < min_SAD:
                    min_SAD = cur_SAD
                    best_point = (i,j)
#Termination condition
if step <= 1:
    break
step = step//2
center_start_x,center_start_y=best_point
center_end_x = center_start_x + block_size
center_end_y = center_start_y + block_size
```

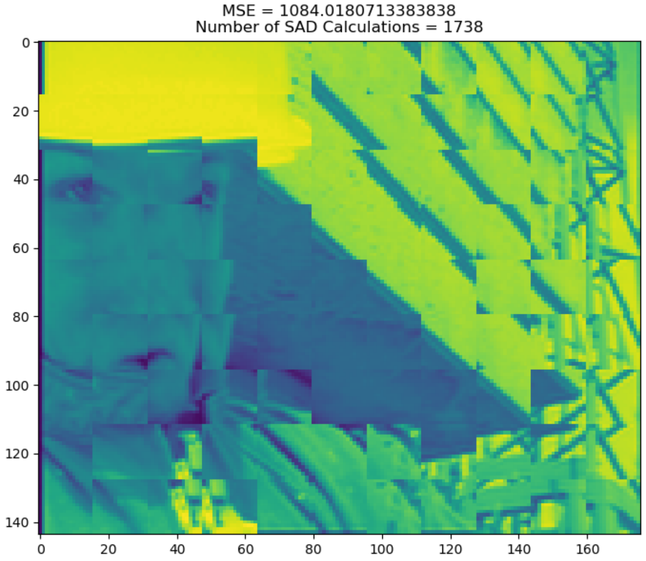
```
        i += 1
    return ((start_x,start_y),(best_point[0]-start_x,best_point[1]-start_y))


#Main Routine Extract the blocks and feed them to the search methods
for i in range(block_size,n_rows+block_size,block_size):
    for j in range(block_size,n_cols+block_size,block_size):
        start_x,end_x,start_y, end_y = i-block_size,i,j-block_size,j
        key,val = EBMA(start_x,end_x,start_y,end_y,frame178,frame179)
        #key,val = two_D_log_search(start_x,end_x,start_y,end_y,frame178,frame179,range_search//2)
        #key,val = three_step_search(start_x,end_x,start_y,end_y,frame178,frame179,range_search//2)
        Optimal_Directions[key]=val


#Construct the Estimated Image
Estimated_Image = np.zeros((n_rows,n_cols))
for i in range(block_size,n_rows+block_size,block_size):
    for j in range(block_size,n_cols+block_size,block_size):
        key = (i-block_size,j-block_size)
        dir_x,dir_y = Optimal_Directions[key]
        Estimated_Image[key[0]:i,key[1]:j]= copy.deepcopy(frame178[key[0]+dir_x:i+dir_x,key[1]+dir_y:j+dir_y])
print('MSE ',MSE(Estimated_Image,frame179),'\nNumber of SAD Calculations',len(SAD_Table))
plt.imshow(Estimated_Image)
plt.show()
```

- **Estimate Image Estimated By EMBA**



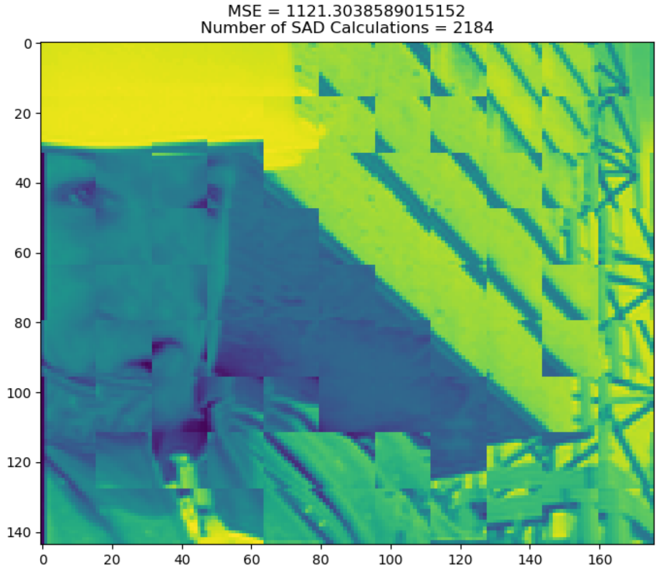
- **MSE = 1080.400**
- **Number of SAD Calculations = 23139**

- **Estimate Image Estimated By 2D Log Search**



- **MSE = 1084.0180**
- **Number of SAD Calculations = 1738**

- **Estimate Image Estimated By Three Step Search**



- **MSE = 1121.3038**
- **Number of SAD Calculations = 2184**