Project Report

By Yukun Duan and Jianlin Dou

2022/11/30

In this project, our group aimed to design a hand gesture-based mouse control system in which we can assess the mouse with gestures without physically touching the mouse. The project's purpose is to detect the hand and count the fingers. When one finger moves horizontally in front of the camera, we can move the mouse, when moving vertically we can left-click. With two fingers detected moving horizontally, we can scroll the mouse wheel, when moving vertically we can right-click.

Of course, this project is pretty challenging for us, we have explored lots of algorithms we learned in class this semester, some of which are already reflected in our code, like motion detection, motion history image, and normal optic flow, while others are dropped because we run into some issues when trying to implement them—for example, KLT feature tracking.

To start, we need to collect data. In this project, we decided to use back cameras from two phones. To implement more conveniently, we set apart one-finger and two-finger situations. Also, we need sights from two directions—the top one can record horizontal movement and the front one can record vertical movement. Therefore, we took four videos, all of which will be saved into different folders.

During the analyze phase, first and for most is the background removal, we split all the videos into single frame and experiment with simple motion detection between consecutive frames using absolute image differencing. And we did this step to all videos recorded.

Then we used the images generated in the first step to detect how many fingers were shown in the video. During this step, we used an algorithm called convex hull to find the fingertip candidate which we called the "farthest point" and the concavities of the hand. What we do is check the angle between these points is within specified limits. Usually, the angle between the tip of our finger and the two closest concavities is within a certain range. The result is below:

卡通人物

中度可信度描述已自动生成卡通人物

中度可信度描述已自动生成

As shown in the lower right corner of the pictures, it correctly calculates the number of fingers.

Next is movement detection. To demonstrate horizontal movement, we used image files from two bird view folders and calculate motion history images which can help us understand the motion location and path as it progresses.  
电脑屏幕的照片

描述已自动生成

As for the click action, we used image files from two front-view folders and implemented the normal flow of fingers in consecutive frames. Because theoretically when the finger moves vertically, MHI will not change on the horizontal level. We also set a threshold, when movement on the negative y-axis exceeds a certain threshold, it will report a click event. Due to some time limitation issues, we didn’t do the release part, but the principle is very similar to “click”, with the direction will be positive rather than negative.

卡通人物

中度可信度描述已自动生成 图片包含 游戏机, 画

描述已自动生成

One of the biggest issues we met is the strict data collection environment, since we manually shot the hand videos. While shotting videos, Hand shaking up and down severely which affect the accuracy of normal optical flow detected result of finger tap action.

Also, our self-made KLT tracking on Harris corner detector detected feature points drift severely, which means tracking along some feature points did not closely follow hand movement correctly. We spend very long time to figure it out. The result is still what it was. It’s not worth to continue, so we give it up.

If we have more time, we consider binding mouse to actually control it. And try some other tracking algorithms, like MeanShift, to track fingertip movement and generate its coordinates to tell mouse the next location to move. And refine normal optical flow motion detection algorithm to better reduce hand shaking effects bring in.

.

Attached Code:

* Background Removal:

import numpy as np  
import cv2  
from matplotlib import pyplot as plt  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 colored\_birdview\_bkg = cv2.imread('birdview/birdview\_bkg.jpg')  
 colored\_side\_bkg = cv2.imread('side/side\_bkg.jpg')  
 birdview\_bkg = cv2.cvtColor(colored\_birdview\_bkg, cv2.COLOR\_BGR2GRAY)  
 side\_bkg = cv2.cvtColor(colored\_side\_bkg, cv2.COLOR\_BGR2GRAY)  
 birdview\_loc = 'birdview/birdview.mp4'  
 side\_loc = 'side/side.mp4'  
 birdview = cv2.VideoCapture(birdview\_loc)  
 side = cv2.VideoCapture(side\_loc)  
 success = True  
 k=1  
 while (success is True):  
 success, frame\_birdview = birdview.read()  
 success2, frame\_side = side.read()  
 success = success and success2  
 if success:  
 gray\_birdview = cv2.cvtColor(frame\_birdview, cv2.COLOR\_BGR2GRAY)  
 gray\_side = cv2.cvtColor(frame\_side, cv2.COLOR\_BGR2GRAY)  
 newbirdview = np.zeros((gray\_birdview.shape[0], gray\_birdview.shape[1]))  
 newside = np.zeros((gray\_side.shape[0], gray\_side.shape[1]))  
 T = 70; #80  
 for i in range(0, newbirdview.shape[0]):  
 for j in range(0, newbirdview.shape[1]):  
 #print("birdview", np.abs(gray\_birdview[i, j] - birdview\_bkg[i, j]))  
 if np.abs(int(gray\_birdview[i, j]) - int(birdview\_bkg[i, j])) > T:  
 newbirdview[i, j] = 1;  
 else:  
 newbirdview[i, j] = 0;  
 for i in range(0, newside.shape[0]):  
 for j in range(0, newside.shape[1]):  
 #print("side", np.abs(gray\_side[i, j] - side\_bkg[i, j]))  
 if np.abs(int(gray\_side[i, j]) - int(side\_bkg[i, j])) > T:  
 newside[i, j] = 1;  
 else:  
 newside[i, j] = 0;  
  
 cv2.imwrite('birdview/birdview\_frame('+str(k)+').jpg', newbirdview\*255)  
 cv2.imwrite('side/side\_frame(' + str(k) + ').jpg', newside\*255)  
 k=k+1  
 print("conversion done!")

* Hand Detector:

import math

import os

import numpy as np

import cv2

from matplotlib import pyplot as plt

def getcnthull(mask\_img):

contours, hierarchy = cv2.findContours(mask\_img, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

contours = max(contours, key=lambda x: cv2.contourArea(x))

epsilon = 0.0085 \* cv2.arcLength(contours, True)

approx = cv2.approxPolyDP(contours, epsilon, True)

hull = cv2.convexHull(approx)

return approx, hull

def getdefects(contours):

hull = cv2.convexHull(contours, returnPoints=False)

defects = cv2.convexityDefects(contours, hull)

return defects

if \_\_name\_\_ == '\_\_main\_\_':

count = 0

k = 1

for i in range(1, len(os.listdir('birdview/')) - 1):

img = cv2.imread('birdview//birdview\_frame('+ str(i) + ').jpg', 0)

kernel1 = np.ones((5, 5), np.uint8)

kernel2 = np.ones((5, 5), np.uint8)

kernel3 = np.ones((4, 4), np.uint8)

img\_birdview = cv2.dilate(cv2.erode(cv2.dilate(img, kernel1, iterations = 1), kernel2, iterations=3), kernel3, iterations = 2)

birdview\_contours, birdview\_hull = getcnthull(img\_birdview)

#cv2.drawContours(img\_birdview, [birdview\_contours], -1, (255, 255, 255), 2)

#cv2.drawContours(img\_birdview, [birdview\_hull], -1, (255, 255, 255), 2)

birdview\_defects = getdefects(birdview\_contours)

if birdview\_defects is not None:

cnt = 0

for i in range(birdview\_defects.shape[0]):

s, e, f, d = birdview\_defects[i][0]

start = tuple(birdview\_contours[s][0])

end = tuple(birdview\_contours[e][0])

far = tuple(birdview\_contours[f][0])

a = np.sqrt((end[0] - start[0]) \*\* 2 + (end[1] - start[1]) \*\* 2)

b = np.sqrt((far[0] - start[0]) \*\* 2 + (far[1] - start[1]) \*\* 2)

c = np.sqrt((end[0] - far[0]) \*\* 2 + (end[1] - far[1]) \*\* 2)

length = (a + b + c) / 2

arc = math.sqrt(length \* (length - a) \* (length - b) \* (length - c))

distance = (2 \* arc) / a

angle = math.acos((b \*\* 2 + c \*\* 2 - a \*\* 2) / (2 \* b \* c))

if angle <= 90 and distance > 230:

cnt += 1

cv2.circle(img\_birdview, far, 6, [255, 255, 255], 10)

cnt += 1

if cnt > 0:

cv2.putText(img, str(cnt), (1206, 681), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE\_AA)#1206\*681

else:

cv2.putText(img, str(0), (1206, 681), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE\_AA)

# plt.figure("finger count img")

# plt.imshow(img, cmap='gray')

# plt.axis('off')

# plt.show()

cv2.imwrite('finger\_counts//birdview\_frame\_with\_finger\_counts('+ str(k) + ').jpg', img)

k += 1

finger\_count = 1 #changable

if cnt == finger\_count:

count = count + 1;

print('The total number of finger equals to ' + str(finger\_count) + ' is: ', count)

cv2.destroyAllWindows()

* Motion History Image Detection:

from skimage.io import imread

from skimage.filters import gaussian

import numpy as np

from matplotlib import pyplot as plt

from skimage import img\_as\_float

import math

aer\_cube = []

for i in range(1,115):

tmp = imread('./test/birdview\_frame('+str(i)+').jpg')

tmp = img\_as\_float(tmp)

aer\_cube.append(tmp)

aer\_cube=np.array(aer\_cube)

f, axarr = plt.subplots(19,6, sharex='col', sharey='row', dpi=400, figsize=(15,15))

for idx in range(0, 113):

axarr[int(idx/6), idx%6].axis('off')

axarr[int(idx/6), idx%6].imshow(np.abs(aer\_cube[idx]-aer\_cube[idx+1]) > 0.08, cmap = 'gray')

aer\_cube.shape

aer\_cube[0].max()

T = aer\_cube.shape[0]

from skimage.morphology import closing

from skimage.morphology import square

aer\_diff = np.abs(aer\_cube[1:,:,:] - aer\_cube[:-1,:,:])

for i in range(aer\_diff.shape[0]):

aer\_diff[i] = closing(aer\_diff[i], square(5)) > 0.08

MHI = np.zeros(aer\_cube.shape)

for idx in range(1, T-1):

for i in range(0, aer\_cube.shape[1]):

for j in range(0, aer\_cube.shape[2]):

if aer\_diff[idx][i][j] == 1:

MHI[idx][i][j] = T

else:

MHI[idx][i][j] = max(0, MHI[idx-1][i][j] - 1)

f, axarr = plt.subplots(19,6, sharex='col', sharey='row', dpi=400, figsize=(15,15))

for idx in range(0, 113):

axarr[int(idx/6), idx%6].axis('off')

axarr[int(idx/6), idx%6].imshow(MHI[idx], cmap = 'gray')

delta\_T = (T-15)/(8)

MHI\_deltaT = np.zeros(MHI.shape)

for idx in range(1, T-1):

for i in range(0, aer\_cube.shape[1]):

for j in range(0, aer\_cube.shape[2]):

if MHI[idx][i][j] - delta\_T > 0:

MHI\_deltaT[idx][i][j] = MHI[idx][i][j] - delta\_T

f, axarr = plt.subplots(19,6, sharex='col', sharey='row', dpi=400, figsize=(15,15))

for idx in range(0, 113):

axarr[int(idx/6), idx%6].axis('off')

axarr[int(idx/6), idx%6].imshow(MHI\_deltaT[idx], cmap = 'gray')

lastMHI = MHI\_deltaT[-2]

lastMHI = lastMHI/T

plt.imshow(lastMHI, cmap='gray')

* Optic Flow Detection:

import math

import os

import cv2

import numpy as np

from matplotlib import pyplot as plt

def normal\_flow(I0, I1):

Gx = np.array([[-1, 0, 1],

[-2, 0, 2],

[-1, 0, 1]]) / 8

Gy = np.transpose(Gx)

avg = np.array([[1, 1, 1],

[1, 1, 1],

[1, 1, 1]]) / 9

fx = cv2.filter2D(I1.astype(np.float32), -1, Gx)

fy = cv2.filter2D(I1.astype(np.float32), -1, Gy)

avg0 = cv2.filter2D(I0.astype(np.float32), -1, avg)

avg1 = cv2.filter2D(I1.astype(np.float32), -1, avg)

ft = avg1 - avg0

x = fx/np.sqrt((np.square(fx) + np.square(fy)))

x = np.nan\_to\_num(x)

y = fy / np.sqrt((np.square(fx) + np.square(fy)))

y = np.nan\_to\_num(y)

magnitudes = ft / np.sqrt(np.square(fx) + np.square(fy))

magnitudes = np.nan\_to\_num(magnitudes)

x\_dir = x\*magnitudes

y\_dir = y\*magnitudes

cv2.normalize(x\_dir, x\_dir, 1, 0, cv2.NORM\_L1)

cv2.normalize(y\_dir, y\_dir, 1, 0, cv2.NORM\_L1)

total\_x = np.sum(x\_dir)

total\_y = np.sum(y\_dir)

# print(x[np.where(x>0)[0], np.where(x>0)[1]])

# total\_x = np.linalg.norm(x\_dir)

#total\_y = np.linalg.norm(y\_dir)

# x\_temp = x\_dir[np.where(x\_dir != 0)[0], np.where(x\_dir != 0)[1]]

# y\_temp = y\_dir[np.where(x\_dir != 0)[0], np.where(x\_dir != 0)[1]]

# if x\_temp.shape[0] > y\_temp.shape[0]:

# y\_temp = np.insert(y\_temp, len(y\_temp), [0]\*(x\_temp.shape[0]-y\_temp.shape[0]))

# elif y\_temp.shape[0] > x\_temp.shape[0]:

# x\_temp = np.insert(x\_temp, len(x\_temp), [0]\*(y\_temp.shape[0]-x\_temp.shape[0]))

# print(x\_dir[np.where(x\_dir != 0)[0], np.where(x\_dir != 0)[1]].shape)

# print(y\_dir[np.where(x\_dir != 0)[0], np.where(x\_dir != 0)[1]].shape)

# print(x\_temp.shape)

# print(y\_temp.shape)

# print(total\_x)

print(total\_y)

# plt.quiver(total\_x, total\_y, color='b', units='xy', scale=1)

# plt.axis('equal')

# plt.xticks(range(-1, 2))

# plt.yticks(range(-1, 2))

# plt.grid()

# plt.show()

return total\_y

if \_\_name\_\_ == '\_\_main\_\_':

threshold = 0.4 #changable

previous\_y\_vector = 0

for i in range(1, len(os.listdir('side\_1/')) - 2):

img1 = cv2.imread('side\_1//side\_frame(' + str(i) + ').jpg', 0)

img2 = cv2.imread('side\_1//side\_frame(' + str(i+1) + ').jpg', 0)

y\_vector = normal\_flow(img1, img2)

if y\_vector < 0 and abs(y\_vector) > threshold:

# a = i+1

# img2 = cv2.imread('side//side\_frame(' + str(a) + ').jpg', 0)

# img3 = cv2.imread('side//side\_frame(' + str(a + 1) + ').jpg', 0)

# y\_vector\_next = normal\_flow(img2, img3)

print('the click operation should happened around Frame(' + str(i) + ')')

cv2.putText(img1, 'clicked', (1150, 681), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE\_AA)

cv2.imwrite('side\_1//side\_frame(' + str(i) + ').jpg', img1)

break

# plt.imshow(img1, cmap='copper')

# plt.axis('off')

# plt.show()