Final project: Hand Tracking & Performing UI actions with CV

آرمین رستمی - سامان نهاوندی پور - سعید بیات

هدف از این پروژه استفاده از تکنیک های بینایی ماشین برای کنترل کردن ماوس و نیز اسکرول کردن با استفاده از حرکت دست می باشد. ابتدا توابع استفاده شده توضیح داده می شوند و درنهایت از کنار هم قرار گیری آنها هدف نهایی حاصل می شود.

تابع زیر یک فریم را که در واقع یک تصویر است به عنوان ورودی دریافت می کند و ابتدا آنرا به فضای رنگی hsv برده و سپس هیستوگرام نرمال شده بخش داخل کادر را باز می گرداند. علت بردن به فضای hsv این است که در این فضا می توان رنگ و روشنایی را جدا کرد و و فقط از رنگ تصویر استفاده کرد.

In [ ]:
```
def get_histogram(frame):
    rows, cols = frame.shape[0], frame.shape[1]
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    irow = int(ROW_START * rows)
    icol = int(COL_START * cols)
    hsv = hsv[irow : irow + ROWS_LEN, icol : icol + COLS_LEN, :]

    hist = cv2.calcHist(
        [hsv], channels=[0, 1], mask=None, histSize=[HUE_MAX, L], ranges=[
    )
    return cv2.normalize(hist, hist, 0, L - 1, cv2.NORM_MINMAX)
```

تابع زیر یک فریم و هیستوگرام ایجاد شده توسط تابع قبلی را دریافت می کند و یک probability image را با توجه به مقادیر رنگی داخل فریم ایجاد می کند. سپس تغییرات لازم در حوزه مکان را به آن اعمال می کند و آنرا به عنوان ماسکی روی فریم ورودی اعمال می کند که درنهایت خروجی آن یک تصویر ماسک شده است که در بخش هایی که رنگ آن مشابه هیستوگرام نیست مقدار آن صفر می باشد.

In [ ]:
```
def get_mask(frame, hist):
    hsv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    proj = cv2.calcBackProject(
        [hsv_frame], channels=[0, 1], hist=hist, ranges=[0, HUE_MAX, 0, L]
    )
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (29, 29))
    proj = cv2.filter2D(proj, -1, kernel)

    proj = cv2.threshold(proj, 150, L - 1, cv2.THRESH_BINARY)[1]

    kernel = np.ones((5, 5), np.uint8)
    proj = cv2.morphologyEx(proj, cv2.MORPH_CLOSE, kernel, iterations=5)

    proj_3d = cv2.merge((proj, proj, proj))
    mask = cv2.bitwise_and(frame, proj_3d)

    return mask
```

تابع زیر تصویر ماسک شده را دریافت می کند و بزرگترین کانتور آن را باز می گرداند.

In [ ]:
```
def get_largest_cnt(mask):
    gray_Mask = cv2.cvtColor(mask, cv2.COLOR_BGR2GRAY)
    thresh = cv2.threshold(gray_Mask, 0, L - 1, 0)[1]
    contours = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SI
    if len(contours) == 0:
        return None
    return max(contours, key=cv2.contourArea)
```

تابع زیر مختصات اولین نقطه بزرگترین کانتور را پس از کاهش نویز باز می گرداند.

In [ ]:
```
def get_coordinates(mask):
    largestCnt = get_largest_cnt(mask)
    if largestCnt is None:
        return None
    coordinates = largestCnt[0][0]
    return reduce_noise(coordinates)
```

تابع زیر مقدار نقطه فعلی را با مقدار قبلی مقایسه می کند و در صورتی که اختلاف آن با مقدار قبلی از حد مشخصی کمتر باشد همان مقدار قبلی را باز می گرداند که باعث کاهش نویز نقاط می شود.

In [ ]:
```
def reduce_noise(coordinates):
    tolerance = 10
    if len(motionPath) > 0:
        lastPoint = motionPath[-1]
        lastCol, lastRow = lastPoint[0], lastPoint[1]
        coordinates[0] = lastCol if abs(coordinates[0] - lastCol) < tolera
        coordinates[1] = lastRow if abs(coordinates[1] - lastRow) < tolera
    return coordinates
```

تابع زیر مختصات نقطه ای را دریافت کرده و آن را به مسیر حرکت دست اضافه می کند که در کاهش نویز و اسکرول کردن استفاده می شود.

In [ ]:
```
def add_to_path(coordinates):
    maxSize = 2
    if len(motionPath) >= maxSize:
        motionPath.pop(0)
    motionPath.append(coordinates)
```

تکه کد زیر از یک کتابخانه پایتون استفاده می کند و توابعی را برای جابجا کردن ماوس و نیز اسکرول کردن پیاده سازی می کند.

```
import pyautogui

pyautogui.PAUSE = 0  # seconds to pause after function calls
pyautogui.FAILSAFE = False  # allows the mouse to exit the window


def move_mouse(coordinates, shape):
    screenCols, screenRows = pyautogui.size()
    col, row = coordinates[0], coordinates[1]
    pyautogui.moveTo(col * screenCols / shape[1], row * screenRows / shape


def scroll(path):
    if len(path) <= 1:
        return
    distance = path[-1][1] - path[-2][1]
    pyautogui.scroll(-distance / 2)
```

:[ ] In

تابع زیر با توجه به مود انتخابی ماوس را جابجا می کند و یا اسکرول می کند.

```
def perform_action(coordinates, shape, mouseAction):
    if mouseAction:
        move_mouse(coordinates, shape)
    else:
        scroll(motionPath)
```

:[ ] In

تابع زیر برای هر فریم داده شده ماسک تصویر را دریافت و نمایش می دهد و سپس مختصات نقطه را دریافت کرده و با توجه به مود انتخابی اکشن مورد نظر را انتخاب می کند.

```
def run(action, hist, frame):
    mask = get_mask(frame, hist)
    cv2.imshow("Mask", mask)
    coordinates = get_coordinates(mask)
    if coordinates is not None:
        add_to_path(coordinates)
        perform_action(coordinates, frame.shape, action)
```

:[ ] In

تابع زیر مربعی که از آن برای کالیبره کردن رنگ ها استفاده می شود را مشخص می کند.

```
def draw_sampling_rectangle(frame):
    rows, cols = frame.shape[0], frame.shape[1]
    irow, icol = int(ROW_START * rows), int(COL_START * cols)
    blue = (255, 0, 0)
    thickness = 3
    cv2.rectangle(frame, (icol, irow), (icol + COLS_LEN, irow + ROWS_LEN),
```

:[ ] In

حال توابع فوق را کنار هم قرار می دهیم تا به هدف نهایی خود برسیم:

ابتدا import کردن توابع مورد نیاز:

In [ ]:
```
from actions import move_mouse, scroll
import numpy as np
import cv2
cv2.__version__
```

مشخص کردن ثوابت:

In [ ]:
```
HUE_MAX = 180
L = 256
ROWS_LEN = 20
COLS_LEN = 20
ROW_START = 0.5
COL_START = 0.8
```

کد درایور برنامه:

فریمی از وبکم دریافت می شود و برنامه منتظر فشرده شدن کلیدی میماند.

سپس کاربر باید دست خود را در داخل کادر کالیبراسیون قرار دهد و دکمه C را فشار دهد. از دکمه های M و S برای تغییر مود بین اسکرول و ماوس استفاده می شود. در صورت فشردن X نیز برنامه متوقف می شود.

اگر برنامه کالیبره شده باشد تابع run که در بالا آمده است فراخوانی می شود. و اگر کالیبره نشده باشد کادری رسم می شود که باید کاربر دست خود را در آن قرار داده و دکمه C را فشار دهد.

```
def pressed(key, expected):
    return key == ord(expected)

motionPath = []
calibrated = False
mouseAction = True
window = cv2.VideoCapture(0)
print(
    """press:
    (C) to calibrate (move your hand to the blue rectangle)
    (S) to switch to scroll mode
    (M) to switch to mouse mode
    (X) to exit\n"""
)
while True:

    frame = cv2.flip(window.read()[1], 1)  # flip the image because it is
    key = cv2.waitKey(1)

    if pressed(key, "x"):
        print("X pressed. exiting...")
        cv2.destroyAllWindows()
        break
    elif pressed(key, "c"):
        hist = get_histogram(frame)
        calibrated = True
        print("Calibration complete. Moving mouse with hand motions...")
    elif pressed(key, "m"):
        mouseAction = True
        print("Switched to mouse mode.")
    elif pressed(key, "s"):
        mouseAction = False
        print("Switched to scroll mode.")

    if calibrated:
        cv2.destroyWindow("Window")
        run(mouseAction, hist, frame)
    else:
        draw_sampling_rectangle(frame)
        cv2.imshow("Window", frame)
```

:[ ] In