Contents

1 object_movement.py

```
#Import packages
from collections import deque
import numpy as np
import argparse
import imutils
import cv2
import pyautogui as pgui
from skimage import exposure
import glob
from cvlib import *
import os

ap = argparse.ArgumentParser()
ap.add_argument("-v", "--video", help = "path to the (optional vide
ap.add_argument("-b", "--buffer", type=int, default=32,
                help="max buffer size")
args = vars(ap.parse_args())

#Create trackbars
cv2.namedWindow('window')

cv2.createTrackbar('H_MAX', 'window', 0, 180, nothing) #H
cv2.createTrackbar('H_MIN', 'window', 0, 180, nothing)

cv2.createTrackbar('S_MAX', 'window', 0, 255, nothing) #S
cv2.createTrackbar('S_MIN', 'window', 0, 255, nothing)

cv2.createTrackbar('V_MAX', 'window', 0, 255, nothing) #V
cv2.createTrackbar('V_MIN', 'window', 0, 255, nothing)

cv2.createTrackbar('Sigma', 'window', 0, 100, nothing)

#Lower and upper HSV boundaries for pink
pinkUpper = (180, 244, 255)
pinkLower = (150, 0, 255)

yellowUpper = (45, 255, 255)
yellowLower = (18, 20, 150)

orangeUpper = (180, 255, 255)
orangeLower = (160, 0, 0)

ref_images = {}
```

```
#load in reference images
for filename in glob.glob(os.path.join('./ref_images', '*.jpg')):
    im = cv2.imread(filename, 0)
    (head, tail) = os.path.split(filename)
    name = tail.split(".")[0]
    print(name)
    im = cv2.flip(im, 1)
    (ret, thresh) = cv2.threshold(im, 127, 255, cv2.THRESH_BINARY)
    ref_images[name] = imutils.resize(thresh, width=600, height=400

#list of tracked points, frame counter, coordinate deltas
pts = deque(maxlen=args["buffer"])
counter = 0
(dx, dy) = (0, 0)
direction = ""
(avgX, avgY) = (10, 10)


#if no video supplied, get the webcam
if not args.get("video", False):
    camera = cv2.VideoCapture(0)

else:
    camera = cv2.VideoCapture(args["video"])

#loop over video frames
while True:
    (grabbed, frame) = camera.read()
    #If a frame wasn't grabbed and this is a video, the video is do
    if args.get("video") and not grabbed:
        break

    #resize the frame, blur it, convert it to HSV
    frame = imutils.resize(frame, height=400)
    ratio = frame.shape[0] / 400
    frame = cv2.flip(frame, 1)
    orig = frame.copy()
    blurred = cv2.GaussianBlur(frame, (11, 11), 0)

    #find contours in mask and get the center of the objects
    colorMask = colorFinder(frame, avgX, avgY, pts, args)

        #grayScale
    gray = cv2.cvtColor(frame.copy(), cv2.COLOR_BGR2GRAY)
```

```
        grayFiltered = cv2.bilateralFilter(gray, 11, 17, 17)
        edges = auto_canny(grayFiltered, cv2.getTrackbarPos('Sigma', 'w

        screenCnt = get_rect(edges)


        warp = warp_perspective(orig, screenCnt, ratio, 600, 400)

        if warp is not None:
            #cv2.imshow("Warp", warp)
            warp = cv2.cvtColor(warp, cv2.COLOR_BGR2GRAY)
            warp = exposure.rescale_intensity(warp, out_range = (0, 255
            (ret, thresh) = cv2.threshold(warp, 127, 255, cv2.THRESH_BI
            #cv2.imshow("threshold", thresh)
            if not thresh is None:
                for key, value in ref_images.items():
                    diff_frame = cv2.bitwise_xor(thresh, value)
                    cv2.imshow("diff", diff_frame)
                    nzCount = cv2.countNonZero(diff_frame)
                    if nzCount < 30000:
                        cv2.putText(frame, key, (10, 60), cv2.FONT_HERS
                            0.65, (0, 0, 255), 3)

        #print(mouseX, mouseY)
        #pgui.moveTo(mouseX, mouseY, 0.0)

        #show our frame
        cv2.imshow("Frame", frame)
        cv2.imshow("Mask", colorMask)
        cv2.imshow("Edges", edges)

        key = cv2.waitKey(1) & 0xFF
        counter += 1

        #if 'q' is pressed, stop the loop
        if key == ord("q"):
            break

camera.release()
cv2.destroyAllWindows()
```

2 cvlib.py

```
from collections import deque
import numpy as np
import argparse
import imutils
import cv2


def nothing(x):
    pass

#Find area of certain color
def colorFinder(frame, avgX, avgY, pts, args):

    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    lower = (cv2.getTrackbarPos('H_MIN', 'window'),
             cv2.getTrackbarPos('S_MIN', 'window'),
             cv2.getTrackbarPos('V_MIN', 'window'))

    upper = (cv2.getTrackbarPos('H_MAX', 'window'),
             cv2.getTrackbarPos('S_MAX', 'window'),
             cv2.getTrackbarPos('V_MAX', 'window'))


    #construct mask for the color then perform
    #dilations and erosions to remove fragment
    mask = cv2.inRange(hsv, lower, upper)
    #mask = cv2.inRange(hsv, yellowLower, yellowUpper)
    #mask = cv2.inRange(hsv, orangeLower, orangeUpper)
    mask = cv2.erode(mask, None, iterations=2)
    mask = cv2.dilate(mask, None, iterations=2)


    cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE)[-2]
    center = None

    if len(cnts) > 0:
        #find the largest contour
        c = max(cnts, key=cv2.contourArea)
        ((x, y), radius) = cv2.minEnclosingCircle(c)
        M = cv2.moments(c)
        center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]
```

```
            if radius > 20:
                #draw the circle
                cv2.circle(frame, (int(x), int(y)), int(radius),
                    (0, 255, 255), 2)
                cv2.circle(frame, center, 5, (0, 0, 255, -1))
                pts.appendleft(center)

        if len(pts) >= args["buffer"] - 1 and pts[-10] is not None:
            avgX = int((pts[-10][0] + pts[0][0] + pts[5][0]) / 3)
            avgY = int((pts[-10][1] + pts[0][1] + pts[5][1]) / 3)

        cv2.putText(frame, "X:{}, Y:{}".format(avgX, avgY), (10, 30), c
            0.65, (0, 0, 255), 3)

        return mask


#Image edge detection
def auto_canny(gray, sigma=0.33):
    #compute median of single channel pixel intensities
    v = np.median(gray)

    lower = int(max(0, (1.0 - sigma) * v))
    upper = int(min(255, (1.0 + sigma) * v))
    edged = cv2.Canny(gray, lower, upper)

    #return the edged image
    return edged

#Find the largest rectangle contour in the image
def get_rect(image):
    cnts = cv2.findContours(image.copy(), cv2.RETR_LIST,
        cv2.CHAIN_APPROX_SIMPLE)[-2]
    cnts = sorted(cnts, key = cv2.contourArea, reverse = True)[:1]
    center = None

    screenCnt = None

    if len(cnts) > 0:
        #Find the rectangle
        for c in cnts:
            #approximate the contours
            peri = cv2.arcLength(c, True)
            approx = cv2.approxPolyDP(c, 0.02 * peri, True)
```

```
            #If our approximated contour has 4 pts, then it's a rec
            if len(approx) == 4:
                screenCnt = approx

                return screenCnt

    return None

#Use matrix math to convert a warped rectangle to one that is seen
def warp_perspective(image, screenCnt, ratio, finalWidth = -1, fina
    if screenCnt is not None:
        #cv2.drawContours(image, [screenCnt], -1, (0, 255, 0), 3)
        #now to identify corners in order to reshape image
        pts = screenCnt.reshape(4, 2)
        rect = np.zeros((4, 2), dtype = "float32")

        s = pts.sum(axis = 1)
        rect[0] = pts[np.argmin(s)] #top-left point
        rect[2] = pts[np.argmax(s)] #top-right point

        #compute difference between points
        diff = np.diff(pts, axis = 1)
        rect[1] = pts[np.argmin(diff)]
        rect[3] = pts[np.argmax(diff)]

        #scale the rectangle to the original image (not scaled down
        rect *= ratio
        (maxWidth, maxHeight) = (0, 0)

        if finalWidth == -1 and finalHeight == -1:
            #get the four corners of the rectangle
            (tl, tr, br, bl) = rect

            #Get the distance between the bottom corners
            widthA = np.sqrt(((br[0] - bl[0]) ** 2) + ((br[1] - bl[

            #Get the distance between the top corners
            widthB = np.sqrt(((tr[0] - tl[0]) ** 2) + ((br[1] - bl[

            #Get the height of the right side
            heightA = np.sqrt(((tr[0] - br[0]) ** 2) + ((tr[1] - br

            #Get the height of the left side
            heightB = np.sqrt(((tl[0] - bl[0]) ** 2) + ((tl[1] - bl
```

```
            #Get the largest height and width bounds
            maxWidth = max(int(widthA), int(widthB))
            maxHeight = max(int(heightA), int(heightB))
        else:
            maxWidth = finalWidth
            maxHeight = finalHeight


        #construct destination points used to map the screen to a t
        dst = np.array([
            [0, 0],       #top-left
            [maxWidth - 1, 0],  #top-right
            [maxWidth - 1, maxHeight - 1],  #bottom-right
            [0, maxHeight - 1]], dtype = "float32") #bottom-left

        #calculate perspective transform matrix and warp perspectiv
        M = cv2.getPerspectiveTransform(rect, dst)
        warp = cv2.warpPerspective(image, M, (maxWidth, maxHeight))

        return warp
    return None
```