# Motion-Code-Presentation

November 13, 2018

## 1 Motion Detection

**Authors:** Metin Senturk Janam Dalal Pooja Umathe
**Date:** 11.13.2018
**Project:** Applications and Methods of Motion Detection

```
In [ ]: import cv2
        import pandas as pd
        import numpy as np
```

### 1.1 Background Subtraction

the main motion detection logic by using the differences btw two frames. function returns difference map.

```
In [ ]: def dist_map(frame1, frame2):
            """outputs pythagorean distance between two frames"""
            frame1_32 = np.float32(frame1)
            frame2_32 = np.float32(frame2)
            diff32 = frame1_32 - frame2_32
            norm32 = np.sqrt(diff32[:, :, 0]**2 + diff32[:, :, 1] **
                             2 + diff32[:, :, 2]**2)/np.sqrt(255**2 + 255**2 + 255**2)
            dist = np.uint8(norm32*255)
            return dist
```

loading face and eye template data into system.
function to detect face and eyes in any given frame.
detecting face and eye. The data that is used.

```
In [ ]: face_cascade, eyes_cascade = init_face_detection()
```

main loop to detect motion. all background steps are covered.

```
In [ ]: video = cv2.VideoCapture(0)
        video.set(3, 1920)
        video.set(4, 1080)

        fourcc = cv2.VideoWriter_fourcc(*'MJPG')
```

```
out = cv2.VideoWriter('output_m.avi',fourcc, 4, (1920, 1080))

while True:
    check, frame = video.read()
    check2, frame2 = video.read()

    motion = 0

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    gray = cv2.GaussianBlur(gray, (21, 21), 0)
    edge = cv2.Canny(gray, 35, 125)

    gray2 = cv2.cvtColor(frame2, cv2.COLOR_BGR2GRAY)
    gray2 = cv2.GaussianBlur(gray, (21, 21), 0)

    #diff_btw_background = cv2.absdiff(static_background, gray)
    diff_btw_background = dist_map(frame, frame2)
    # diff_btw_background = diff_img(static_background, gray, gray2)

    threshold_frame = cv2.threshold(
        diff_btw_background, 30, 255, cv2.THRESH_BINARY)[1]
    threshold_frame = cv2.dilate(threshold_frame, None, iterations=2)

    (_, cnts, _) = cv2.findContours(
        threshold_frame.copy(),
        cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE
    )

    for countour in cnts:
        if cv2.contourArea(countour) < 10000:
            continue
        else:
            (x, y, w, h) = cv2.boundingRect(countour)
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 3)

    frame = face_detect(frame)


    out.write(frame)

    cv2.imshow("frame", frame)
    cv2.imshow("diff_btw_background:", diff_btw_background)
    cv2.imshow("threshold_frame:", threshold_frame)

    if cv2.waitKey(1) == ord('q'):
        break
```

releasing all resources.

```
In [ ]: video.release()
        out.release()
        cv2.destroyAllWindows()
```

### 1.2 Meanshift

constructing meanshift detection

```
In [ ]: # setup initial location of window
        r, h, c, w = 250, 90, 400, 125  # simply hardcoded the values
        track_window = (c, r, w, h)
        # set up the ROI for tracking
        roi = frame[r:r+h, c:c+w]
        hsv_roi = cv.cvtColor(roi, cv.COLOR_BGR2HSV)
        mask = cv.inRange(hsv_roi, np.array((0., 60., 32.)),
                                np.array((180., 255., 255.)))
        roi_hist = cv.calcHist([hsv_roi], [0], mask, [180], [0, 180])
        cv.normalize(roi_hist, roi_hist, 0, 255, cv.NORM_MINMAX)
        # Setup the termination criteria, either 10 iteration or move by atleast 1 pt
        term_crit = (cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT, 10, 1)
```

process of meanshift object detection

```
In [ ]: while(1):
                ret, frame = cap.read()
                if ret == True:
                    hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
                    dst = cv.calcBackProject([hsv], [0], roi_hist, [0, 180], 1)
                    # apply meanshift to get the new location
                    ret, track_window = cv.CamShift(dst, track_window, term_crit)
                    # Draw it on image
                    x, y, w, h = track_window
                    img2 = cv.rectangle(frame, (x, y), (x+w, y+h), 255, 2)

                    # write the flipped frame
                    out.write(img2)

                    cv.imshow('img2', img2)

                    if cv.waitKey(60) & 0xff == ord('q'):
                        break
                else:
                    break
```

### 1.3 Face Motion Tracking and Detection

```
In [ ]: def init_face_detection():
            face_cascade = cv2.CascadeClassifier()
            eyes_cascade = cv2.CascadeClassifier()

            if not face_cascade.load('../data/haarcascade_frontalface_default.xml'):
                print('--(!)Error loading face cascade')
                exit(0)
            if not eyes_cascade.load('../data/haarcascade_eye.xml'):
                print('--(!)Error loading eyes cascade')
                exit(0)

            return face_cascade, eyes_cascade
```

```
In [ ]: def face_detect(frame):
            frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            frame_gray = cv2.equalizeHist(frame_gray)
            # -- Detect faces
            faces = face_cascade.detectMultiScale(
                frame_gray,
                scaleFactor=1.1,
                minNeighbors=5,
                minSize=(30, 30)
            )
            for (x, y, w, h) in faces:
                center = (x + w//2, y + h//2)
                frame = cv2.ellipse(frame, center, (w//2, h//2),
                                   0, 0, 360, (255, 0, 255), 4)
                faceROI = frame_gray[y:y+h, x:x+w]

                # -- In each face, detect eyes
                eyes = eyes_cascade.detectMultiScale(faceROI)
                for (x2, y2, w2, h2) in eyes:
                    eye_center = (x + x2 + w2//2, y + y2 + h2//2)
                    radius = int(round((w2 + h2)*0.25))
                    frame = cv2.circle(frame, eye_center, radius, (255, 0, 0), 4)

            return frame
```

### 1.4 Hand Motion Detection and Tracking

function to detect the hand motion and fingers

```
In [ ]: def calculateFingers(res, drawing):
            #  convexity defect
            hull = cv2.convexHull(res, returnPoints=False)
            if len(hull) > 3:
                defects = cv2.convexityDefects(res, hull)
                if defects is not None:
                    cnt = 0
                    for i in range(defects.shape[0]):  # calculate the angle
```

```
                        s, e, f, d = defects[i][0]
                        start = tuple(res[s][0])
                        end = tuple(res[e][0])
                        far = tuple(res[f][0])
                        a = math.sqrt((end[0] - start[0]) ** 2 + (end[1] - start[1]) ** 2)
                        b = math.sqrt((far[0] - start[0]) ** 2 + (far[1] - start[1]) ** 2)
                        c = math.sqrt((end[0] - far[0]) ** 2 + (end[1] - far[1]) ** 2)
                        angle = math.acos((b ** 2 + c ** 2 - a ** 2) / (2 * b * c))  # cosine th
                        if angle <= math.pi / 2:  # angle less than 90 degree, treat as fingers
                            cnt += 1
                            cv2.circle(drawing, far, 8, [211, 84, 0], -1)
                    if cnt > 0:
                        return True, cnt+1
                    else:
                        return True, 0
            return False, 0
```

the process of hand gesture tracking

```
In [ ]: while camera.isOpened():
            #Main Camera
            ret, frame = camera.read()
            frame = cv2.bilateralFilter(frame, 5, 50, 100)  # Smoothing
            frame = cv2.flip(frame, 1)  #Horizontal Flip
            cv2.imshow('original', frame)


            #Background Removal
            bgModel = cv2.createBackgroundSubtractorMOG2(0, 50)
            fgmask = bgModel.apply(frame)

            kernel = np.ones((3, 3), np.uint8)
            fgmask = cv2.erode(fgmask, kernel, iterations=1)
            img = cv2.bitwise_and(frame, frame, mask=fgmask)

            # Skin detect and thresholding
            hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
            lower = np.array([0, 48, 80], dtype="uint8")
            upper = np.array([20, 255, 255], dtype="uint8")
            skinMask = cv2.inRange(hsv, lower, upper)
            cv2.imshow('Threshold Hands', skinMask)

            # Getting the contours and convex hull
            skinMask1 = copy.deepcopy(skinMask)
            _,contours, hierarchy = cv2.findContours(skinMask1, cv2.RETR_TREE, cv2.CHAIN_APPROX_
            length = len(contours)
            maxArea = -1
            if length > 0:
```

```
    for i in range(length):
        temp = contours[i]
        area = cv2.contourArea(temp)
        if area > maxArea:
            maxArea = area
            ci = i

    res = contours[ci]
    hull = cv2.convexHull(res)
    drawing = np.zeros(img.shape, np.uint8)
    cv2.drawContours(drawing, [res], 0, (0, 255, 0), 2)
    cv2.drawContours(drawing, [hull], 0, (0, 0, 255), 3)

    isFinishCal, cnt = calculateFingers(res, drawing)
    print("Fingers"), cnt
    cv2.imshow('output', drawing)

outf.write(frame)
out.write(drawing)

if cv2.waitKey(1) == ord('q'):
    break
```