CHAPTER-7

IMPLEMENTATION WITH SOURCE CODE

7.1 T-rex_game.py

```
# -*- coding: utf-8 -*-
"""
Created on Sun Feb 28 17:05:41 2019

@author: PAWAN
"""
import cv2
import pyautogui as mouse
model=cv2.CascadeClassifier('closed_palm.xml')
video=cv2.VideoCapture(0)
while True:
    ret,frames=video.read()
    objects=model.detectMultiScale(frames,1.2,10)
    if len(objects):
        mouse.press('space')
    cv2.imshow('Video',frames)
    if cv2.waitKey(1)&0xFF==ord('q'):
        break
video.release()
cv2.destroyAllWindows()
```

7.2 OpenCV.py

```
# -*- coding: utf-8 -*-
"""
Created on Sun Feb 24 17:05:41 2019

@author: PAWAN
"""
import sys
```

```
import os
import pyautogui
import cv2
import numpy as np
import math

cap = cv2.VideoCapture(0)
while(cap.isOpened()):
    # read image
    ret, img = cap.read()
    cv2.imwrite('opencv.png',img)

    # get hand data from the rectangle sub window on the screen
    cv2.rectangle(img, (300,300), (100,100), (0,255,0),0)
    crop_img = img[100:300, 100:300]

    # convert to grayscale
    grey = cv2.cvtColor(crop_img, cv2.COLOR_BGR2GRAY)

    # applying gaussian blur
    value = (35, 35)
    blurred = cv2.GaussianBlur(grey, value, 0)

    # thresholdin: Otsu's Binarization method
    _, thresh1 = cv2.threshold(blurred, 127, 255,
                               cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)

    # show thresholded image
    cv2.imshow('Thresholded', thresh1)

    # check OpenCV version to avoid unpacking error
    (version, _, _) = cv2.__version__.split('.')

    if version == '3':
        image, contours, hierarchy = cv2.findContours(thresh1.copy(), \
```

```
            cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
    elif version == '2':
        contours, hierarchy = cv2.findContours(thresh1.copy(),cv2.RETR_TREE, \
            cv2.CHAIN_APPROX_NONE)

    # find contour with max area
    cnt = max(contours, key = lambda x: cv2.contourArea(x))

    # create bounding rectangle around the contour (can skip below two lines)
    x, y, w, h = cv2.boundingRect(cnt)
    cv2.rectangle(crop_img, (x, y), (x+w, y+h), (0, 0, 255), 0)

    # finding convex hull
    hull = cv2.convexHull(cnt)

    # drawing contours
    drawing = np.zeros(crop_img.shape,np.uint8)
    cv2.drawContours(drawing, [cnt], 0, (0, 255, 0), 0)
    cv2.drawContours(drawing, [hull], 0,(0, 0, 255), 0)

    # finding convex hull
    hull = cv2.convexHull(cnt, returnPoints=False)

    # finding convexity defects
    defects = cv2.convexityDefects(cnt, hull)
    count_defects = 0
    cv2.drawContours(thresh1, contours, -1, (0, 255, 0), 3)

    # applying Cosine Rule to find angle for all defects (between fingers)
    # with angle > 90 degrees and ignore defects
    for i in range(defects.shape[0]):
        s,e,f,d = defects[i,0]

        start = tuple(cnt[s][0])
        end = tuple(cnt[e][0])
```

```
        far = tuple(cnt[f][0])

        # find length of all sides of triangle
        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)

        # apply cosine rule here
        angle = math.acos((b**2 + c**2 - a**2)/(2*b*c)) * 57

        # ignore angles > 90 and highlight rest with red dots
        if angle <= 90:
            count_defects += 1
            cv2.circle(crop_img, far, 1, [0,0,255], -1)
        #dist = cv2.pointPolygonTest(cnt,far,True)

        # draw a line from start to end i.e. the convex points (finger tips)
        # (can skip this part)
        cv2.line(crop_img,start, end, [0,255,0], 2)
        #cv2.circle(crop_img,far,5,[0,0,255],-1)

    # define actions required
    if count_defects == 1:

        cv2.putText(img, "Message 2", (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 2, 2)
        cv2.imwrite('opencv2.png',img)
    elif count_defects == 2:
        #pyautogui.click(288,403)
        #pyautogui.hotkey("command","up")
        cv2.putText(img, "Message 3", (5, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, 2)
    elif count_defects == 3:
        cv2.putText(img, "Message 4", (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 2, 2)
    elif count_defects == 4:
        cv2.putText(img, "Message 5", (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 2, 2)
        #os.startfile(r"C:\Users\91990\Desktop\FInal_year_project\project
```

```
files\basics.docx")
        #sys.exit()
        #cv2.putText(img, "Message 4", (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 2, 2)
    else:
        cv2.putText(img,"Default Message", (50, 50),\
                cv2.FONT_HERSHEY_SIMPLEX, 2, 2)

    # show appropriate images in windows
    cv2.imshow('Basic Hand Gestures', img)
    all_img = np.hstack((drawing, crop_img))
    cv2.imshow('Contours', all_img)

    #k = cv2.waitKey(10)
    if cv2.waitKey(1) == ord('a'):
        cap.release()
        cv2.destroyAllWindows()
        break
```

7.3 main.py

```
import keyboard
import cv2
import numpy as np
import math
import os
import time
import pyautogui
from point import p_crop
from hand import h_crop
from fin import f_crop
#from fist import fs_crop
from thumbdown import t_crop
from okay import ok_crop
```

```
os.chdir("C:/Users/91990/Desktop/Final year project/Hand-Gesture-Recognition-for-Presentation-Process-in-Python-master")
h1_cascade=cv2.CascadeClassifier('hand.xml')
okay_cascade = cv2.CascadeClassifier('ok.xml')
point_cascade = cv2.CascadeClassifier('point1.xml')
fin_cascade=cv2.CascadeClassifier('fin_2.xml')
fist_cascade=cv2.CascadeClassifier('fist.xml')
thumbdown_cascade = cv2.CascadeClassifier('thumbdown.xml')
cap = cv2.VideoCapture(0)
ca=0

while(1):
    _,img=cap.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    point=point_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=3,flags=0, minSize=(100,80))
    fin=fin_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=3,flags=0, minSize=(100,80))
    hand=h1_cascade.detectMultiScale(gray,1.1, 5)
    fist=fist_cascade.detectMultiScale(gray,1.3, 5)
    okay=okay_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=3,flags=0, minSize=(100,150))
    thumbdown=thumbdown_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=3,flags=0, minSize=(100,80))

    if fist is not ():
       pyautogui.hotkey("command", "up" )



    for (x,y,w,h) in okay:
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,0,255),2)
        roi_gray=gray[y:y+h,x:x+w]
        roi_color=img[y:y+h,x:x+w]
        crop_img=img[y:y+h,x:x+w]
        ok_crop(crop_img,img)
```

```
        #time.sleep(0.5)


    for (x,y,w,h) in point:
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,255),2)
        roi_gray=gray[y:y+h,x:x+w]
        roi_color=img[y:y+h,x:x+w]
        crop_img=img[y:y+h,x:x+w]
        ha=2
        p_crop(crop_img,img)
        #time.sleep(0.5)

    for (x,y,w,h) in fin:
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,255,0),2)
        roi_gray=gray[y:y+h,x:x+w]
        roi_color=img[y:y+h,x:x+w]
        crop_img=img[y:y+h,x:x+w]
        f_crop(crop_img,img)
        #time.sleep(0.5)


    for (x,y,w,h) in thumbdown:
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
        roi_gray=gray[y:y+h,x:x+w]
        roi_color=img[y:y+h,x:x+w]
        crop_img=img[y:y+h,x:x+w]
        ha=5
        t_crop(crop_img,img)
        #time.sleep(0.5)

    for (x,y,w,h) in hand:
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
        roi_gray=gray[y:y+h,x:x+w]
        roi_color=img[y:y+h,x:x+w]
        crop_img=img[y:y+h,x:x+w]
```

```
            h_crop(crop_img,img)

        for (x,y,w,h) in fist:
            cv2.rectangle(img, (x,y), (x+w,y+h), (127,0,255), 2)
            roi_gray=gray[y:y+h,x:x+w]
            roi_color=img[y:y+h,x:x+w]
            crop_img=img[y:y+h,x:x+w]
            h_crop(crop_img,img)
        cv2.imshow('Feed',img)

        if cv2.waitKey(1) == ord('a'):
            cap.release()
            cv2.destroyAllWindows()
            break
```

7.4 point.py

```
import keyboard
import cv2
import numpy as np
import math
import os
import time
import pyautogui
os.chdir("C:/Users/91990/Desktop/Final year project/Hand-Gesture-Recognition-for-Presentation-Process-in-Python-master")
def p_crop(crop_img,img):
    grey = cv2.cvtColor(crop_img, cv2.COLOR_BGR2GRAY)

    # applying gaussian blur
    value = (35, 35)
    blurred = cv2.GaussianBlur(grey, value, 0)

    # thresholdin: Otsu's Binarization method
    _, thresh1 = cv2.threshold(blurred, 127, 255,
```

```
                cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)

    # show thresholded image
    cv2.imshow('Thresholded', thresh1)

    # check OpenCV version to avoid unpacking error
    (version, _, _) = cv2.__version__.split('.')

    if version == '3':
        image, contours, hierarchy = cv2.findContours(thresh1.copy(), \
            cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
    elif version == '2':
        contours, hierarchy = cv2.findContours(thresh1.copy(),cv2.RETR_TREE, \
            cv2.CHAIN_APPROX_NONE)

    # find contour with max area
    cnt = max(contours, key = lambda x: cv2.contourArea(x))

    # create bounding rectangle around the contour (can skip below two lines)
    x, y, w, h = cv2.boundingRect(cnt)
    cv2.rectangle(crop_img, (x, y), (x+w, y+h), (0, 0, 255), 0)

    # finding convex hull
    hull = cv2.convexHull(cnt)

    # drawing contours
    drawing = np.zeros(crop_img.shape,np.uint8)
    cv2.drawContours(drawing, [cnt], 0, (0, 255, 0), 0)
    cv2.drawContours(drawing, [hull], 0,(0, 0, 255), 0)

    # finding convex hull
    hull = cv2.convexHull(cnt, returnPoints=False)

    # finding convexity defects
    defects = cv2.convexityDefects(cnt, hull)
```

```
    count_defects = 0
    cv2.drawContours(thresh1, contours, -1, (0, 255, 0), 3)

    # applying Cosine Rule to find angle for all defects (between fingers)
    # with angle > 90 degrees and ignore defects
    for i in range(defects.shape[0]):
        s,e,f,d = defects[i,0]

        start = tuple(cnt[s][0])
        end = tuple(cnt[e][0])
        far = tuple(cnt[f][0])
        if d>10000:
            count_defects+=1

    # define actions required
    count_defects+=1
    if count_defects == 2:
        #keyboard.press_and_release('left')
        pyautogui.hotkey("command", "left" )
    else:
        cv2.putText(img,"0", (50, 50),\
            cv2.FONT_HERSHEY_SIMPLEX, 2, 2)
    ha=0

    #time.sleep(1)
    # show appropriate images in windows
    #cv2.imshow('Gesture', img)
    all_img = np.hstack((drawing, crop_img))
    cv2.imshow('Contours', all_img)
```

7.5 fist.py

```
import keyboard
import cv2
import numpy as np
```

```
import math
import os
import time
import pyautogui
os.chdir("C:/Users/91990/Desktop/Final year project/Hand-Gesture-Recognition-for-Presentation-Process-in-Python-master")
def fs_crop(crop_img,img):
    grey = cv2.cvtColor(crop_img, cv2.COLOR_BGR2GRAY)

    # applying gaussian blur
    value = (35, 35)
    blurred = cv2.GaussianBlur(grey, value, 0)

    # thresholdin: Otsu's Binarization method
    _, thresh1 = cv2.threshold(blurred, 127, 255,
                               cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)

    # show thresholded image
    cv2.imshow('Thresholded', thresh1)

    # check OpenCV version to avoid unpacking error
    (version, _, _) = cv2.__version__.split('.')

    if version == '3':
        image, contours, hierarchy = cv2.findContours(thresh1.copy(), \
               cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
    elif version == '2':
        contours, hierarchy = cv2.findContours(thresh1.copy(),cv2.RETR_TREE, \
               cv2.CHAIN_APPROX_NONE)

    # find contour with max area
    cnt = max(contours, key = lambda x: cv2.contourArea(x))

    # create bounding rectangle around the contour (can skip below two lines)
    x, y, w, h = cv2.boundingRect(cnt)
```

```
cv2.rectangle(crop_img, (x, y), (x+w, y+h), (0, 0, 255), 0)

# finding convex hull
hull = cv2.convexHull(cnt)

# drawing contours
drawing = np.zeros(crop_img.shape,np.uint8)
cv2.drawContours(drawing, [cnt], 0, (0, 255, 0), 0)
cv2.drawContours(drawing, [hull], 0,(0, 0, 255), 0)

# finding convex hull
hull = cv2.convexHull(cnt, returnPoints=False)

# finding convexity defects
defects = cv2.convexityDefects(cnt, hull)
count_defects = 0
cv2.drawContours(thresh1, contours, -1, (0, 255, 0), 3)

# applying Cosine Rule to find angle for all defects (between fingers)
# with angle > 90 degrees and ignore defects
for i in range(defects.shape[0]):
    s,e,f,d = defects[i,0]

    start = tuple(cnt[s][0])
    end = tuple(cnt[e][0])
    far = tuple(cnt[f][0])
    if d>10000:
        count_defects+=1

# define actions required

count_defects+=1
# define actions required
if count_defects == 1:
    cv2.putText(img,"ha", (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 2, 2)
```

```
        #keyboard.press_and_release('enter')
        pyautogui.hotkey("command", "up" )
    else:
        cv2.putText(img,"0", (50, 50),\
            cv2.FONT_HERSHEY_SIMPLEX, 2, 2)
    ha=0

    #time.sleep(1)
    # show appropriate images in windows
    #cv2.imshow('Gesture', img)
    all_img = np.hstack((drawing, crop_img))
    cv2.imshow('Contours', all_img)
```

7.6 thumbdown.py

```
import keyboard
import cv2
import numpy as np
import math
import os
import time
import pyautogui
os.chdir("C:/Users/91990/Desktop/Final year project/Hand-Gesture-Recognition-for-Presentation-Process-in-Python-master")
def t_crop(crop_img,img):
    grey = cv2.cvtColor(crop_img, cv2.COLOR_BGR2GRAY)

    # applying gaussian blur
    value = (35, 35)
    blurred = cv2.GaussianBlur(grey, value, 0)

    # thresholdin: Otsu's Binarization method
    _, thresh1 = cv2.threshold(blurred, 127, 255,
                               cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
```

```
# show thresholded image
cv2.imshow('Thresholded', thresh1)

# check OpenCV version to avoid unpacking error
(version, _, _) = cv2.__version__.split('.')

if version == '3':
    image, contours, hierarchy = cv2.findContours(thresh1.copy(), \
        cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
elif version == '2':
    contours, hierarchy = cv2.findContours(thresh1.copy(),cv2.RETR_TREE, \
        cv2.CHAIN_APPROX_NONE)

# find contour with max area
cnt = max(contours, key = lambda x: cv2.contourArea(x))

# create bounding rectangle around the contour (can skip below two lines)
x, y, w, h = cv2.boundingRect(cnt)
cv2.rectangle(crop_img, (x, y), (x+w, y+h), (0, 0, 255), 0)

# finding convex hull
hull = cv2.convexHull(cnt)

# drawing contours
drawing = np.zeros(crop_img.shape,np.uint8)
cv2.drawContours(drawing, [cnt], 0, (0, 255, 0), 0)
cv2.drawContours(drawing, [hull], 0,(0, 0, 255), 0)

# finding convex hull
hull = cv2.convexHull(cnt, returnPoints=False)

# finding convexity defects
defects = cv2.convexityDefects(cnt, hull)
count_defects = 0
cv2.drawContours(thresh1, contours, -1, (0, 255, 0), 3)
```

```
# applying Cosine Rule to find angle for all defects (between fingers)
# with angle > 90 degrees and ignore defects
for i in range(defects.shape[0]):
    s,e,f,d = defects[i,0]

    start = tuple(cnt[s][0])
    end = tuple(cnt[e][0])
    far = tuple(cnt[f][0])
    if d>9000:
        count_defects+=1
    # find length of all sides of triangle
    # define actions required
count_defects+=1
if count_defects==2 or count_defects==1:
    #keyboard.press_and_release('enter')
    pyautogui.hotkey("command", "up" )

else:
    cv2.putText(img,"cmd down", (50, 50),\
        cv2.FONT_HERSHEY_SIMPLEX, 2, 2)
ha=0
# drawing contours
drawing = np.zeros(crop_img.shape,np.uint8)
cv2.drawContours(drawing, [cnt], 0, (0, 255, 0), 0)
cv2.drawContours(drawing, [hull], 0,(0, 0, 255), 0)

# finding convex hull
hull = cv2.convexHull(cnt, returnPoints=False)

# finding convexity defects
defects = cv2.convexityDefects(cnt, hull)
count_defects = 0
cv2.drawContours(thresh1, contours, -1, (0, 255, 0), 3)
```

```
#time.sleep(1)
# show appropriate images in windows
#cv2.imshow('Gesture', img)
all_img = np.hstack((drawing, crop_img))
cv2.imshow('Contours', all_img)
```