Aerial Robotics Kharagpur Task Round Documentation
Task 2 : FACE DETECTION GAME

Nikhil Giri*

***Abstract*— This Documentation gives a summary of how I approached task 2 given to me during ARK Software Selection Round**

From the time when I was first given these tasks to the time i am writing this documentation I have learned and explored a lot of things. I learnt a lots of Path finding algorithms and also tried implementing them. These algorithms are essential for drones to judge and correct their path on basis of algorithms they encounter.

In this task we create a simple ball player game with just a twist that instead of keyboard player object is controlled by the face of the user. We have created this game using Image Processing. we used Face detection , Masking and Code logic.

I. INTRODUCTION

Some of the drones use image processing continuously to identify objects in their surroundings. This task easily demonstrates the difficulties of it. Along with problem such as lighting, this task also forces us to come up with a optimal solution to process each image as fast as possible. In Task 2 we have to create a game using Image processing Where the player object is positioned and moved according to the position and movement of the User's Face in front of Webcam. This player object have to keep bouncing a ball.

II. PROBLEM STATEMENT

In this task we have to create a game environment.In this game environment where user can play a game. Player's face will determine the player position. For that we have to detect and track the user face. In this process we assume that the user face is a perfect circle. Next we have to create a ball element which obeys laws of physics in a gravity less environment. The ball has co-efficient of elasticity as one so it bounces perfectly with the top, left and right walls. The player character when comes in contact with the ball, we assume a perfect collision and the ball bounces off. But if the player misses and the ball drops to the bottom wall(Floor) then it is Game over.

The task can be spilt into following Sub-Tasks :

1) Detecting and tracking of the user's face
2) Creating the Game environment and coding the ball physics
3) Integrating the Face detection output into the game environment
4) Setting the Logic for Collision and the new path of ball after collision
5) Bug Fixing
6) Additional functionality such as replay and a score system

In task the major issue is to keep the back-end calculation and processes as low as possible as we have to operate those processes about 30 time per second (Due to 30 FPS of my camera)

III. RELATED WORK

I tried to learn advance face tracking methods via packages like mediapipe but did not implement as my final approach was adequate. But I Plan to learn and implement it in future.

IV. INITIAL ATTEMPTS

Initially my approach was two create 2 files. One file would detect the face approximate it to a circle and return a center co-ordinate. The other file will create a game environment and have my ball physics and collision code in it and then I will create a 3rd final python file where i will combine both of them. This sure was a big mistake as not only the merger of the two files gave a lot of issues as the compiler would just execute one file and wait foe it to be over to start the next file. But when i somehow resolved all errors the code lagged very badly. The calculations performed per frame weer two slow which over time created a lag and the game was not at all smooth. To overcome this I thought of scanning only one frame in 10 frames and the game went a bit smooth but this mad another problem which is that if the user moves fast enough it seemed that the player character just teleported so the game was now lagging and also looked bad. So i decided to skip the project then and attempt later. The final approach didn't clicked until a week before submission and led to some all nighters and finally i completed this task via the approach discussed in the final approach section.

V. FINAL APPROACH

The Final Approach was possible due to the concept known as Masking. In the case of this specific task we are using bitwise-and masking.

How the algorithm theoretically works is that the algorithm first waits for the camera to read a frame. As soon as the camera reads an image of the user, the algorithm creates an empty image i.e. an image which is entirely black and of the same size of as that of the frame read by camera (Webcam in this case). Now the face detection algorithm works on the frame and it detects the position of the face. The face detection algorithm then approximate the face as a circle. Now, once we approximate the face as a circle, we have the center co-ordinates and the radius of this approximated circle. Using this two values, we can create a white circle on

the earlier mentioned mask, empty black picture, at the exact same position where the algo detected the face in the Frame. Now that we have Mask and the frame which is essentially the picture captured by our camera. Once we have both of them , we merge them by a process known, as Bitwise AND masking. This process will make all the pixel in the frame which correspond to black pixel in mask will be converted as black else the pixel corresponding white are left unchanged.

Now, then we apply our game logic on it, and then we again, read for the camera to read a new frame and create a new mask.

By using this concept. Now we have an game environment, where we have the face object already inside it without any major pixel wise operation. We don't need a function to give output of the center of the face and radius of the face. Rather we have the player object, which is face implanted in the game.All we need is to have ball object and code its game physics,scenarios and other cases. This saves a lot of processing power required and improves the speed of the game and have results into a smooth lag free gameplay.

Now lets see the details and go step wise:

Part 1: Face Tracking The main thing in user control is face detection and accurate detection of the face.Once we do that we have to approximate it as an circle and find out its radius and centre co-ordinates.

```
1
2  face_cascade = cv2.CascadeClassifier('haarcascades/
       haarcascade_frontalface_alt.xml')
3
4  video = cv2.VideoCapture(0)
5
6
7  h0 = 480  #height of the image from my webcam
8  w0 = 640  #width of the image from my webcam
9
10 while True:
11     check, frame = video.read()
12     frame = cv2.flip(frame,1)
13     #print(frame.shape) --> [480,640,3]
14     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
15     faces = face_cascade.detectMultiScale(gray,
       scaleFactor=1.2, minNeighbors=5)
16
17     for (x, y, w, h) in faces:
18         cx =(x + w // 2)
19         cy = y + h // 2
20         radius = h // 2
21         cv2.circle(frame, (cx,cy), radius, (255,
       0,255),2)
```

Listing 1. Detecting and tracking of the user's face

Here, we are essentially using haar cascade detection on the gray scaled image as i found that it works better this way. On top of that since the user is going to react to its position in the game window the image has to be flipped as the laptop webcam flips the image when they record so the player would see himself going in other direction which might became confusing therefore in line 12 i flip the image read by the webcam.

Later we find the image and find the radius and the center co-ordinates to draw a circle around the face.

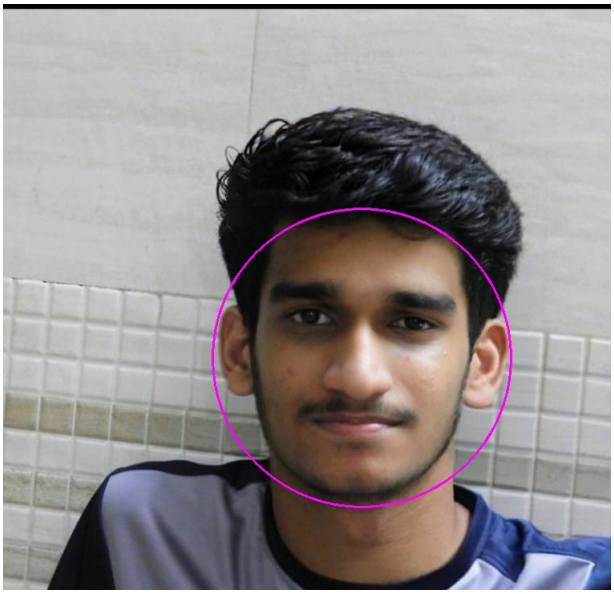
Part 2: Game Environment (Masking in our approach)

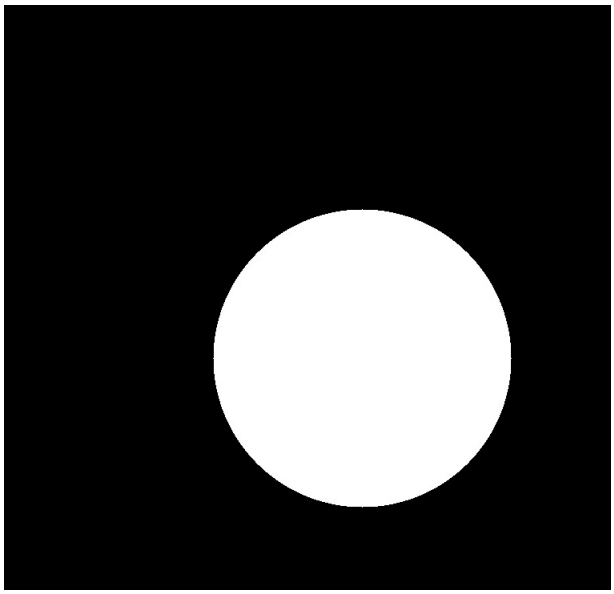
```
1  while True:
2     #**********part1************
3
4      frame = cv2.copyMakeBorder( frame, 100, 100,
       100, 100, cv2.BORDER_CONSTANT,value = 0)
5      mask = np.zeros_like(frame)
6
7      mask = cv2.circle(mask, (cx,cy), radius,
       (255,255,255), -1)
8
9      result = cv2.bitwise_and(frame, mask)
10     cv2.rectangle(result,(100,100),(100+w0,100+h0)
       ,(0,255,255),2)
```

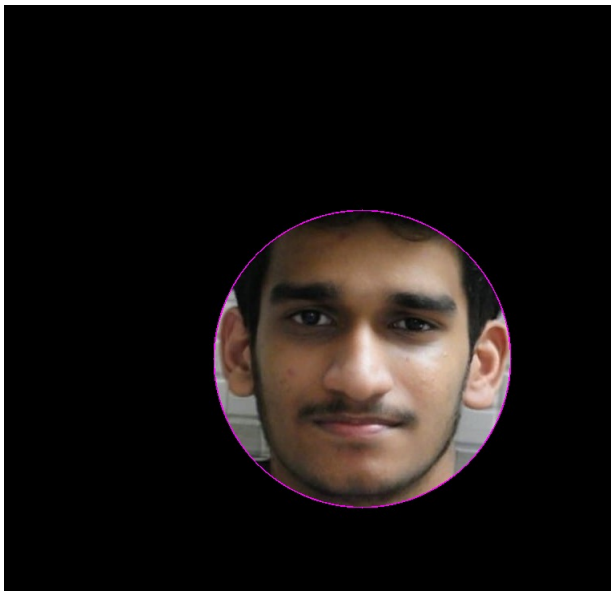
Listing 2. Detecting and tracking of the user's face

In this part of the code we are padding by 100 on all the sides to give the environment extra space other that the gameplay region. Then we create an empty mask of the same shape as that of the padded frame. Now as we discussed we create a white circle in the mask image at the same spot as users face in the original frame considering the padding added. This completes our mask creation for the current frame and now we move on to application of the mask.

Essentially we are doing and operation pixel wise in the frame. The Black pixel refers to 0 or FALSE in the mask and when applied by bitwise And, we get a zero value or black color. White on other hand refers to 255 or True and we get the original values when mask is applied







Part 3: The Ball dynamics

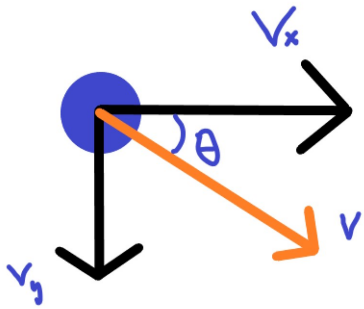
As we can see that we get the player controller already inside the environment and can be easily controlled by the user by moving the head in any direction. Now its time to add the ball to the game environment.

First off we create a rectangle to differentiate between the gamespace and padded region then we initialize the ball at a fixed specific location. Then to add more randomness and challenge to the gave we do not move the ball in 45 degrees rather we generate a random number between 20 and 70 which we use as the angle by which the ball first starts moving.

If theta is the random angle then the dx(Vx) and dy(Vy) velocities will be :

$$V_x = V_o * cos\theta$$

$$V_y = V_o * sin\theta$$



Once we initialize the ball at a position then we need to change the position of the ball in each frame refresh. Remember that the game environment background have to refresh every time stem to make sure we don't form multiple balls.

Thus, if bx and by are the position of the balls :

$$B_x = B_x + d_x$$

$$B_y = B_y + d_y$$

Now we have to compute the wall collision. If the ball collide with the right wall or left wall then the Vx should reverse. If the top wall is collided with then Vy should reverse. By reverse I mean direction change i.e. value multiplied by -1.

Thus if Cx and Cy are the center co-ordinates :

If (Cx + radius = Right limit) or (Cx - radius = left Limit) then $d_x = -1 * d_x$

If (Cy - radius = Top limit) then $d_y = -1 * d_y$

Now lets see the code for the Ball movement Mechanics :

```
h0 = 480  #height of the image from my webcam
w0 = 640  #width of the image from my webcam

cx = 0
cy = 0
radius = 0
speed = 10
angle =  random.randrange(20,70)
dx,dy =int(speed*math.cos(math.radians(angle))),int
    (speed*math.sin(math.radians(angle)))
xb,yb = 120,120

while True:
   #***********PART 1 **************
   #***********PART 2 **************
    result = cv2.bitwise_and(frame, mask)

    # Gameplay border
    cv2.rectangle(result,(100,100),(100+w0,100+h0)
    ,(0,255,255),2)

    #Motion --> New position of the ball
    xb = xb+dx
    yb = yb+dy

    #Drawing the ball at the new position
    cv2.circle(result,(xb,yb),20,(255,0,0),-1)

    #Ball-Wall collision Logic
    if yb<=120 :
        dy *= -1
    if xb >=720 or xb<=120:
```

```
        dx *= -1
```

Listing 3. Ball movement Mechanics

This is a simple code implementing what we just discussed above.

Part 4: Ball- Player Collision Logic Since we have complicated the game by changing the angle at which the ball first starts then we also need to determine the new angle with respect to the axes when the two circular objects collide. For this case we have two sub parts namely, detection of collision and finding and applying the new Dx and Dy.

Detection of collision:

This part is the easy one we will constantly keep on checking the distance between the centre of the ball and centre of the player face. If they equal to the sum radius of the ball and radius of the face object ideally means that they both have collided. In this case we also give some breathing space of a small number so that collisions are detected fast and precisely

Determination of the new angle :

When the ball approaches the player from the left side the new angle comes out to be 2*phi - theta where phi is the angle of inclination of the tangent at point of contact. Also we can find out that the same for approach from right side results in angle of theta - 2*phi

Thus converting both of this into code:

```

while True:
   #***********PART 1 **************
   #***********PART 2 **************
   #***********PART 3 **************
    cb_x = (cx - xb)
    cb_y = (cy - yb)

    dist = math.sqrt(cb_y**2 + cb_x**2)

    if dist<= radius + 20 + 10 and dist > radius +
    3: #if ball is deep then no direction change so
     it goes more deep and reset
        score += 10
        phi = math.atan(cb_y/cb_x)

        if xb <= cx:

            dx = -1*int(speed*(math.cos(-1*math.
    radians(angle) + (2*phi))))
            dy = -1*int(speed*(math.sin(math.
    radians(angle) + (2*phi))))
        else :

            dx = -1*int(speed*(math.cos(1*math.
    radians(angle) - (2*phi))))
            dy = -1*int(speed*(math.sin(1*math.
    radians(angle) - (2*phi))))
```

Listing 4. Ball-Player collision Mechanics

Part 5: Bug fixing

After creating this much , I started observations and found some bugs in the code and attempted to solve them. I am listing some of them here.

.//Too close to screen

In the mentioned program i have allowed the player to roam freely in the game space which does not pose that much of a issue ass the ball initialization is random but there is no restriction on how large the player object can be. In this case the player can sit close to the webcam and easily play the game as it will be easy to manage the ball and chances to miss will be next to none.

To rectify this I placed a restriction on how large the player object can be using the following code:

```
def addText(img,text,position,color =(0,255,0) ):
    cv2.putText(img,text,
            position,
            font,
            fontScale,
            color,
            lineType)
font                   = cv2.FONT_HERSHEY_SIMPLEX
fontScale              = 1
fontColor              = (0,255,0)
lineType               = 2
while True :
    #*****************Part 1,2,3,4
    ******************
    if radius > 120 :
            #print(radius)
            addText(result,"Please move Back"
    ,(300,300))
            cv2.imshow(win, result) # to update the
     image before exit
            cv2.waitKey(20)

        continue
```

Listing 5. Player Radius control

I have selected a value of 120, if the radius is more that this value then there will be a text printed that the player should move back and the game pauses until the player moves back. Once the radius changes to the acceptable value, the game resumes instantly.

.//Player engulfs the ball

Another bug possible is that players starts moving too fast and gets in the new position of the ball before the ball gets displayed this lead the ball to be engulfed inside the player and start bouncing internally. To stop this we need to check if the ball entered the player and if so then we reset the game back to initial position and warn the user to not move so fast

Code implementation:

```
def resetgame():
    xb = 120
    yb = 120
    angle =  random.randrange(20,70)
    dx,dy =int(10*math.cos(math.radians(angle))),
    int(10*math.sin(math.radians(angle)))
    score = 0
    return xb,yb,angle,dx,dy,score

while True :
    #*****************Part 1,2,3,4
    ******************
    if dist< radius +20 - 5 :
        xb,yb,angle,dx,dy,score = resetgame()
        addText(result,"Please move slowly"
    ,(300,300))
        cv2.waitKey(300)
```

Listing 6. Engulfing strategy control

Part 6: Additional Functionality

After playing the game many times I have added many user friendly Functions. In these part I will discuss some of them:

//Start Page

When the code starts compiling the game used to directly start giving the user no time calm down his/her nerves and focus on the game. so i added a Starting page where the game will start only if the user presses 'A' button from the keyboard. Even after pressing 'A', there will be a countdown of 3 seconds and after that the game will begin.

Code implementation :

```
win = "Lets Play a Game"
empty = np.zeros((680,840,3))

while True :

    cv2.imshow(win,empty)
    addText(empty,"Press A to Begin!!!!!!!"
    ,(250,350))

    k = cv2.waitKey(1)
    if k == ord('a'):
        cv2.waitKey(1000)
        for i in range(3):
            empty = np.zeros((680,840,3))

            addText(empty,"Starting in " + str(3-i)
     ,(280,320))
            cv2.imshow(win,empty)
            cv2.waitKey(1000)
            #print("Here")
        break
```

Listing 7. Starting Page

//Restart the game

Usually after the game ends the used would have to re-run the code to play again. To solve this I added a Press 'R' for restart in the game end scenario. If the player does not press any button for 20 Seconds then the game will end by itself. But if the player presses 'R' before this 20 seconds then the game will reset using the reset game function mentioned earlier and the user can play again without hassle.

Code implementation :

```

while True :
    #**************Part 1,2,3,4,5 ***************
    if yb >=560:
        print("Game Over!!!")
        addText(result,"Game Over!!!!!!", (300,300)
    )
        addText(result," Press R to Restart "
    ,(280,330))
        addText(result,"Press Q to Quit", (290,360)
    )
        cv2.imshow(win, result) # to update the
    image before exit
        k = cv2.waitKey(20000)
        if k == ord('q'):
            break
        elif k == ord('r'):
            xb,yb,angle,dx,dy,score = resetgame()
        else:
            #ends game if no response for 20 secs
            break
```

Listing 8. Play again function

VI. Results and Observation

The Result is a decent game which looks like this :

The Game works best when played in a well lit environment, Solid color background and user's face should not be moved very fast.

VII. Future Work

Though I tried to solve as many bugs as possible, At final testing I saw a bug that the ball can bounce multiple times without reaching the wall, increasing the score.. this bug can be easily solved by using to flags to check that the ball-Player collision does not take place in consecutive frames. I would fix this bug and add some more features in the future. I also plan to implement Tkinter Gui and create a full-fledged Game app.

CONCLUSION

This Task was very interesting as we had to imitate physical nature without using physics but rather image processing. Since the game had to be played for a duration rather than a program which runs once and then ends, I found a lots of bugs and had to do lots of debugging . This Task essentially gave me an idea how creating application for different user can be made and tested. The Game must be played in a well lit environment and the users motion must not be very fast. Except some minor issues in some test runs, the game runs well.

References

[1] OpenCV-Python Documentation, OpenCV-Python Tutorials",2021.