Human Pose Estimation (& Replication) Based Game

Project Report

EKLAVYA MENTORSHIP PROGRAMME

At

SOCIETY OF ROBOTICS AND AUTOMATION,
VEERMATA JIJABAI TECHNOLOGICAL INSTITUTE
MUMBAI

JUNE 2020

ACKNOWLEDGMENT

Salutations to the Supreme Nature and Its' Mystical Force.

We are grateful for the help and guidance given by our mentor Mr. Saharsh Jain. We are also grateful to the Society of Robotics and Automation (SRA), VJTI for providing us with platforms such as the Eklavya 2020.

Shreyas Swanand Atre
shreyasatre16@gmail.com

Sarah Nisar Tisekar
sarahtisekar@gmail.com

Dhruvi Doshi
drdoshi29@gmail.com

Arnav Ganatra
arnavganatra@gmail.com

TABLE OF CONTENTS

Project Overview

Surveillance was an important concern and a matter of public security. Computer Vision which is exponentially growing has an elegant approach towards it through various types of Convolutional Neural Networks. Hence began the journey of Human Pose Estimation. It has gained the attention of the Computer Vision community for the past few decades. It is a crucial step towards understanding human actions in images and videos. Pose Determination has myriad applications in real time.We hereby, present an application of Pose Estimation (2D) in Gaming. Where we have used the PoseNet[0] by Google which is one of the most widely used models in Human Pose Estimation.

1.INTRODUCTION

1.1 What is Human Pose Estimation?

In General Human Pose Estimation is (as its name suggests) estimation of key-points i.e. estimation of the locations of joints of a human body.

See Fig 1.1

The number of key-points varies according to the model we train / use.The PoseNet estimates 17 joints of our body. Figure 1.2 Shows the points and their corresponding Ids.

1.2 Why do we need PoseNet ?

Mainly because we do not have huge computation power and storage to train on huge datasets. Besides, PoseNet is extremely lighter on computation and storage. It also offers quantization.

1.3 Are there other models available?

Yes. There are many others. But currently PoseNet has satisfied our needs.

Other pose estimation models include OpenPose[1] which includes Unity plugin and needs CUDA enabled.

Also this[2] is a model ,a pytorch implementation estimates 3D poses.

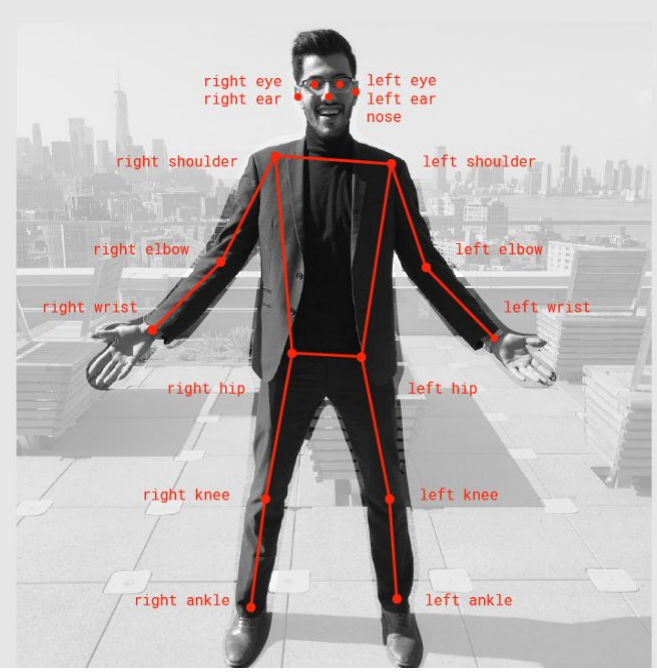


Fig 1.1

Id	Part
0	nose
1	leftEye
2	rightEye
3	leftEar
4	rightEar
5	leftShoulder
6	rightShoulder
7	leftElbow
8	rightElbow
9	leftWrist
10	rightWrist
11	leftHip
12	rightHip
13	leftKnee
14	rightKnee
15	leftAnkle
16	rightAnkle

Fig. 1.2

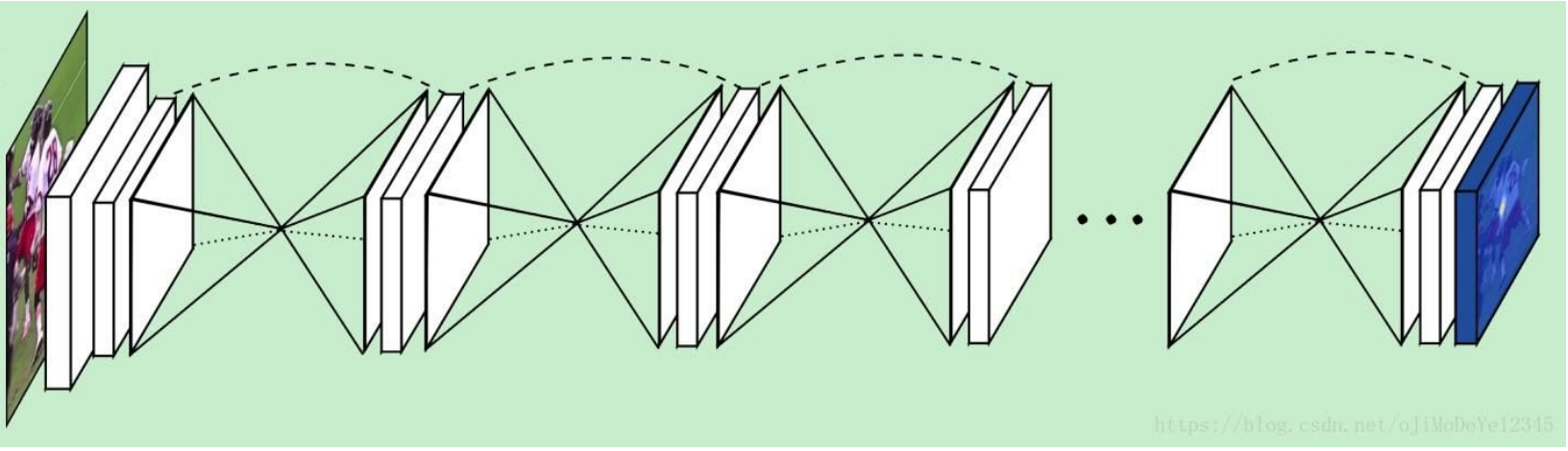
2 In Depth Analysis

2.1 General

To create a ConvNet for the joint-detection process, we need an architecture that extracts the features of human joints and then finally relates those joints. This also solves the problem if any joint is not visible in the image, it would estimate its location. This[3] paper describes the approach thoroughly. There are several other approaches, however Stacked HourGlass is one of the best known.

2.2 Stacked Hourglass Model

As the name Suggests , there are repeated Convolutions, Max-Pooling and then step-ups. This



helps in extracting features at every scale and the results have shown that it stands apart from general approaches. The predictions of each hourglass in the stack are supervised, and not only the final hourglass predictions. Paper[4]

2.3 Bottom-Up Part Based Geometric Embedding

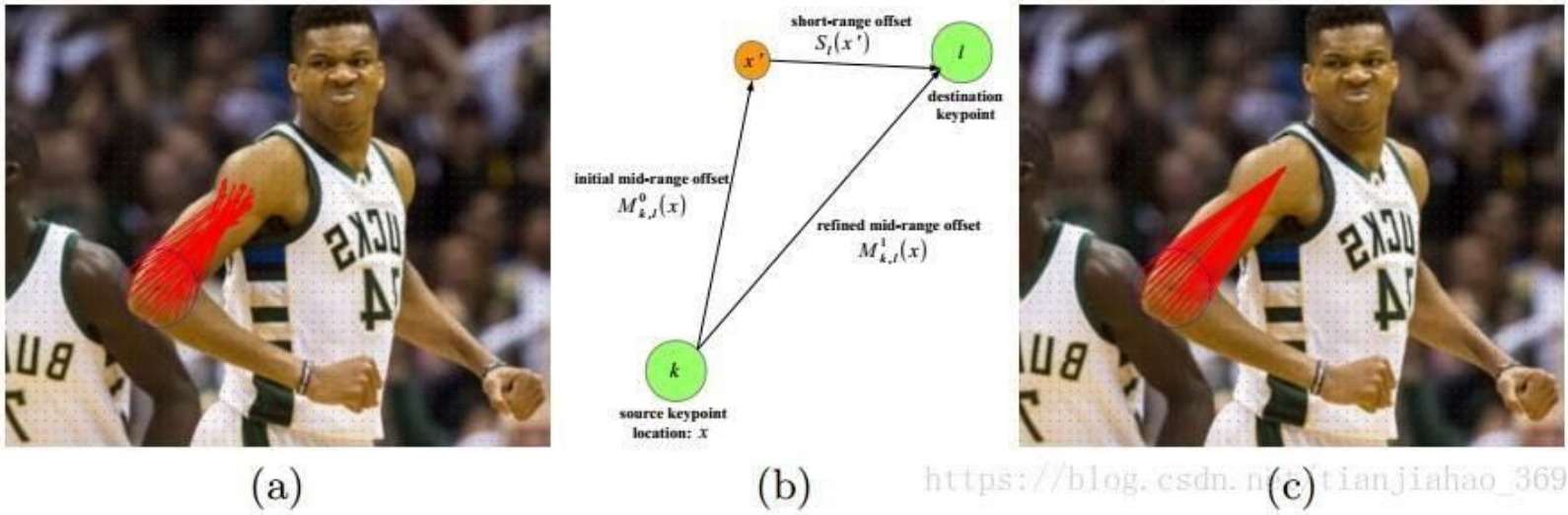
This is one of the approaches that gives a precise position of joints.

Firstly we segment the image into grids, further for each spatial position, we first classify whether it is in the vicinity of each of the K key-points or not (which we call a “heatmap”), then predict a 2-D local offset vector to get a more precise estimate of the corresponding key-point location.

Paper[5]

2.4 Grouping Key-points Obtained

After obtaining the key-points from heatmaps and offset vectors, a displacement vector is obtained from pairwise offsets and gradient losses. See Paper[6] for exact terms and formulae.



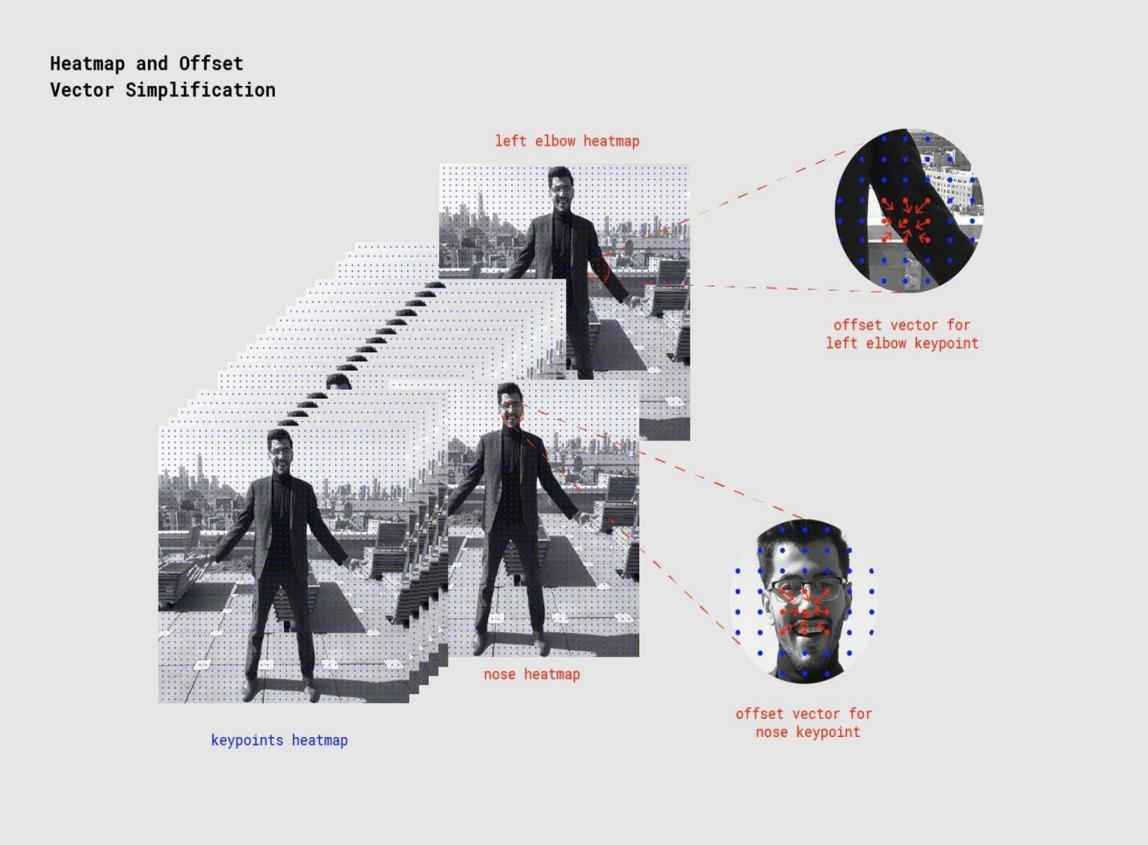
2.5 Handling Input and Output(s)

The input is an RGB image.The PoseNet model gives the output as heatmaps ,offset vectors, displacement vectors.

The PoseNet model is image size invariant, which means it can predict pose positions in the same scale as the original image regardless of whether the image is downscaled.The output stride determines how much we're scaling down the output relative to the input image size.

2.5.1 Model Outputs: Heatmaps and Offset Vectors

When PoseNet processes an image, what is in fact returned is a heatmap along with offset vectors that can be decoded to find high confidence areas in the image that correspond to pose keypoints. The illustration below captures at a high-level how each of the pose keypoints is associated with one heatmap tensor and an offset vector tensor.



2.5.2 Heatmaps

Each heatmap is a 3D tensor of size resolution x resolution x 17, since 17 is the number of keypoints detected by PoseNet. Each position in that heatmap has a confidence score, which is the probability that a part of that keypoint type exists in that position. It can be thought of as the original image being broken up into a 15x15 grid, where the heatmap scores provide a classification of how likely each keypoint exists in each grid square.

2.5.3 Offset Vectors

Each offset vector is a 3D tensor of size resolution x resolution x 34, where 34 is the number of keypoints * 2. Since heatmaps are an approximation of where the keypoints are, the offset vectors correspond in location to the heatmap points, and are used to predict the exact location of the key-points as by traveling along the vector from the corresponding heatmap point.

Estimating Poses from the Outputs of the Model

After the image is fed through the model, we perform a few calculations to estimate the pose from the outputs. The single-pose estimation algorithm for example returns a pose confidence score which itself contains an array of keypoints (indexed by part ID) each with a confidence score and x, y position.

2.5.4 Final Steps

To get the key-points of the pose:
1.A sigmoid activation is done on the heatmap to get the scores.

```
scores = heatmap.sigmoid()
```

2.argmax2d is done on the keypoint confidence scores to get the x and y index in the heatmap with the highest score for each part, which is essentially where the part is most likely to exist. This produces a tensor of size 17x2, with each row being the y and x index in the heatmap with the highest score for each part.

```
heatmapPositions = scores.argmax(y, x)
```

3.The offset vector for each part is retrieved by getting the x and y from the offsets corresponding to the x and y index in the heatmap for that part. This produces a tensor of size 17x2, with each row being the offset vector for the corresponding keypoint. For example, for the part at index k, when the heatmap position is y and d, the offset vector is:

```
offsetVector = [offsets.get(y, x, k), offsets.get(y, x, 17 + k)]
```

4.To get the keypoint, each part's heatmap x and y are multiplied by the output stride then added to their corresponding offset vector, which is in the same scale as the original image.

```
keypointPositions = heatmapPositions * outputStride + offsetVectors
```

5.Finally, each keypoint confidence score is the confidence score of its heatmap position.
The pose confidence score is the mean of the scores of the keypoints.

3. Application

3.1 Integration With Game

To integrate the 2D Pose estimation with the game, we use the 2D key points from Posenet to detect the gestures and accordingly we control the movements of the game character using keyboard controls. We divide the frame into a grid and the location of the lower neck(midpoint of the two shoulder key points) is used to detect the position of the body.

1. Body-centre = Nothing
2. Body-right = character moves right
3. Body-left = character moves left
4. Body-up = jump

For ease of use we add hand controls for the jump. Video Link : [Game Controls Demo]

Code for keyboard control:

Defining grid pattern and location of center of left and right shoulder.

posenet/webcam_demo.py

```
 ################# Defining the grid ############################
 gidd = np.array((args.cam_width//3,args.cam_width//1.5,\
                  args.cam_height//3,args.cam_height//1.5))
horizontal = args.cam_height//2
def grid(center,display_image,row,column,fwd_bwd,right_hand ,left_hand):
  right_hand =np.array(right_hand,dtype=np.int)
  left_hand = np.array(left_hand,dtype=np.int)
  row_copy=row column_copy=column fwd_bwd_copy=fwd_bwd
  if(center[0]>gidd[0] and center[0]<gidd[1]):
    column='center'
  elif(center[0]<gidd[0]):
    column='left'
  if left_hand[0]<= horizontal :
    fwd_bwd = "Backward"
  else :
    fwd_bwd = "Forward"
```

```
############ For overlaying-image ###################################
```

```
 cv2.circle(display_image,(right_hand[1],right_hand[0]),10,(255,255,0),-1)
 cv2.circle(display_image,(left_hand[1],left_hand[0]),10,(255,255,0),-1)
 cv2.circle(display_image,(args.cam_width//2,horizontal),10,(255,255,255),-1)
 cv2.putText(display_image,fwd_bwd,(600,400),cv2.FONT_HERSHEY_SIMPLEX,3,(0,255,255),5,c
v2.LINE_AA)
cv2.putText(display_image, str_no_str, (600,490), cv2.FONT_HERSHEY_SIMPLEX,3,
(0,255,255),      5, cv2.LINE_AA)
 ##########################################################################
```

```
   if(column_copy!=column):
    print('column change')
   if(column_copy=='right'):
    if(column=='center'):
      keyboard.press_and_release('left')
 ####### And similarly we can do it further ##########################
```

To execute the game and the webcam_demo.py file simultaneously, we import the subprocess and threading module.

4 GAME DESCRIPTION

4.1. Description

Game is made using Unity.This game called NotEndlessRunner,has a character in the game which keeps on running on a path laid with obstacles.The character movements include moving to left, right and jump.

4.1.1.User control

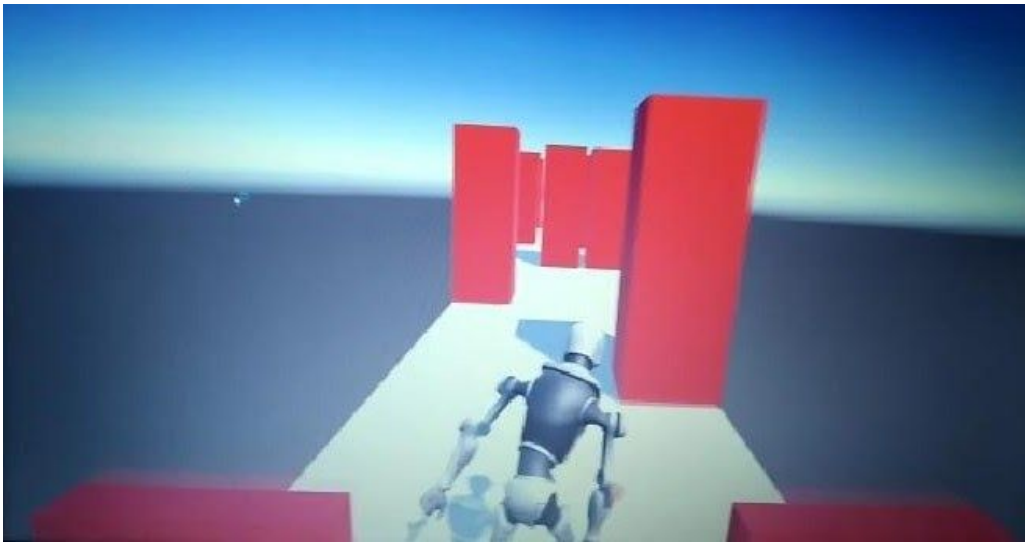
The user can control the character by its gestures.

1.If the user moves right, the character moves right.

2.If the user moves left, the character moves left.

3.For the jump, the user should raise his left hand.

Hence, by using the above controls the user has to save the character from colliding with the obstacles.



4.2. Unity Design Components

A GameObject is an object in Unity Editor which contains components. Components define the behaviour of that GameObject

Common component configurations

This section details some fundamental default component configurations in Unity

1. Transform Component

Every GameObject in Unity has a transform component. This component defines the GameObject's position, rotation, and scale in the game world and Scene view. You cannot remove this component.The Transform component also enables the concept of parenting, which allows you to make a GameObject a child of another GameObject and control its position via the parent's Transform component. This is a fundamental part of working with GameObjects in Unity.

2.Main Camera GameObject components

By default, every new Scene starts with a GameObject called Main Camera. This GameObject is configured to act as the primary camera in your game. It contains the Transform component, the Camera component, and an Audio Listener to pick up audio in your application.

Creating components with scripting

Scripting (or creating scripts) is writing your own additions to the Unity Editor's functionality in code, using the Unity Scripting API. When you create a script and attach it to a GameObject, the script appears in the GameObject's Inspector just like a built-in component.

In technical terms, any script you make compiles as a type of component, so the Unity Editor treats your script like a built-in component. You define the members of the script to be exposed in the Inspector, and the Editor executes whatever functionality you've written.

Below is the code snippet of the CharacterMovementScript from NotEndlessRunner

The name of the class is taken from the name you supplied when the file was created. The class name and file name must be the same to enable the script component to be attached to a GameObject.

```
  using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
public class CharacterMovementScript : MonoBehaviour
{
       private Animator animator;
       private int lane;
       public Transform explodeObj;
```

```
// Start is called before the first frame update
      void Start()
      {
             lane = 0;
             animator = GetComponent<Animator>();
       }
```

```
// Update is called once per frame
      void Update()
      {
             if(Input.GetKeyDown(KeyCode.LeftArrow)
             {
                    if (lane > -1)
                           lane--;
             }
             if(Input.GetKeyDown(KeyCode.RightArrow))
             {
                    if (lane < 1)
                    lane++;
             }
             if(Input.GetKeyDown(KeyCode.Space))
             {
                    animator.SetTrigger("JumpButtonDown");
                    animator.SetTrigger("continueRunning");
             }
             Vector3 newPosition = transform.position;
             newPosition.x = lane;
             transform.position = newPosition;
             transform.Rotate(Vector3.up, 0.0f);
      }
```

```
       void OnCollisionEnter(Collision collided)
      {
             if(collided.gameObject.tag == "Obstruction")
             {
                    numberScript.lvlCompStatus = "fail";
                    Instantiate(explodeObj, transform.position, explodeObj.rotation);
                    Destroy(gameObject);
              }
             if(collided.gameObject.tag == "EndTheGame")
             {
                    numberScript.lvlCompStatus = "success";
              }
       }
}
```

3D Physics

This section describes the main components available through Unity's built-in 3D physics engine, which you can use in object-oriented projects.

1.Rigidbody

A Rigidbody is the main component that enables physical behaviour for a GameObject. With a Rigidbody attached, the object will immediately respond to gravity. If one or more Collider components are also added, the GameObject is moved by incoming collisions.

2.Colliders

Collider components define the shape of a GameObject for the purposes of physical collisions. A collider, which is invisible, does not need to be the exact same shape as the GameObject's mesh. A rough approximation of the mesh is often more efficient and indistinguishable in gameplay.

The simplest (and least processor-intensive) colliders are **primitive** collider types. In 3D, these are

i. Box Collider

Box Colliders are rectangular cuboids and are useful for items such as crates or chests. However, you can use a thin box as a floor, wall or ramp. The Box Collider is also a useful element in a Compound Collider.

ii. Capsule Collider

You can adjust the Capsule Collider's Radius and Height independently of each other. It is used in the Character Controller and works well for poles, or can be combined with other Colliders for unusual shapes.

iii. Sphere Collider

The collider can be resized via the Radius property but cannot be scaled along the three axes independently (ie, you can't flatten the sphere into an ellipse). As well as the obvious use for spherical objects like tennis balls, etc, the sphere also works well for falling boulders and other objects that need to roll and tumble.

Primitive colliders do not work correctly with shear transforms. If you use a combination of rotations and non-uniform scales in the Transform hierarchy so that the resulting shape is no longer a primitive shape, the primitive collider cannot represent it correctly.

Mesh Collider

There are some cases, however, where even compound colliders are not accurate enough. In 3D, you can use Mesh Colliders to match the shape of the GameObject's mesh exactly.These colliders are much more processor-intensive than primitive types.Also, a mesh collider cannot collide with another mesh collider (i.e., nothing happens when they make contact). You can get around this in some cases by marking the mesh collider as **Convex** in the **Inspector**. This generates the collider shape as a "convex hull" which is like the original mesh but with any undercuts filled in.The benefit of this is that a convex mesh collider can collide with other mesh colliders so you can use this feature when you have a moving character with a suitable shape. However, a good rule is to use mesh colliders for **scene** geometry and approximate the shape of moving GameObjects using compound primitive colliders.

Static Colliders

You can add colliders to a GameObject without a Rigidbody component to create floors, walls and other motionless elements of a Scene. These are referred to as *static* colliders. At the opposite, colliders on a GameObject that has a Rigidbody are known as *dynamic* colliders. Static colliders can interact with dynamic colliders but since they don't have a Rigidbody, they don't move in response to collisions.

Triggers

The scripting system can detect when collisions occur and initiate actions using the `OnCollisionEnter` function. However, you can also use the **physics engine** simply to detect when one collider enters the space of another without creating a collision. A collider configured as a **Trigger** (using the **Is Trigger** property) does not behave as a solid object and will simply allow other colliders to pass through. When a collider enters its space, a trigger will call the `OnTriggerEnter` function on the trigger object's **scripts**

Collision callbacks for scripts

When collisions occur, the physics engine calls functions with specific names on any scripts attached to the objects involved. You can place any code you like in these functions to respond to the collision event. For example, you might play a crash sound effect when a car bumps into an obstacle.On the first physics update where the collision is detected, the `OnCollisionEnter` function is called. During updates where contact is maintained, `OnCollisionStay` is called and finally, `OnCollisionExit` indicates that contact has been broken. Trigger colliders call the analogous OnTriggerEnter, OnTriggerStay and `OnTriggerExit` functions.With normal, non-trigger collisions, there is an additional detail that at least one of the objects involved must have a non-kinematic Rigidbody (ie, *Is Kinematic* must be switched off). If both objects are kinematic Rigidbodies then `OnCollisionEnter`, etc, will not be called. With trigger collisions, this restriction doesn't apply and so both kinematic and non-kinematic Rigidbodies will prompt a call to `OnTriggerEnter` when they enter a trigger collider.

.

Animation

Animator Controllers

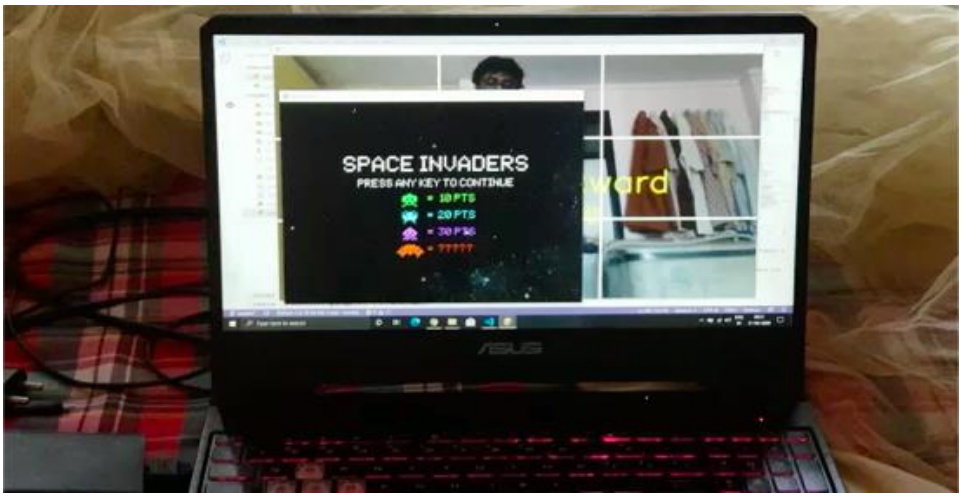
An Animator Controller allows you to arrange and maintain a set of animations for a character or other animated Game Object. The controller has references to the animation clips used within it, and manages the various animation states and the transitions between them using a so-called State Machine, which could be thought of as a kind of flow-chart, or a simple program written in a visual programming language within Unity.

Blend Trees

Blend Trees are used for allowing multiple animations to be blended smoothly by incorporating parts of them all to varying degrees. The amount that each of the motions contributes to the final effect is controlled using a *blending parameter*, which is just one of the numeric animation parameters associated with the Animator Controller. In order for the blended motion to make sense, the motions that are blended must be of similar nature and timing. Blend Trees are a special type of state in an Animation State Machine

4.3. Making this work for other games

One can use this for any games with simpler controls as this[7]



if your game has the same controls as this one,it can be simply achieved by changing the game path in *webcam_demo.py* .

posenet/webcam_demo.py

```
def opengame():
 cmd = "NotEndlessRunner.exe"
```

```
    subprocess.call(cmd, shell=True)
    return None
```

If you are having trouble changing the path ,place the game executable along with its .dll into the same root directory and change the name of the game.exe here and you are ready to play.

On the other hand, if your game has other controls or if you wish to change the gestures,you'll need to make the changes here.

```
if(column_copy!=column):
        # print('column change')
        if(column_copy=='right'):
            if(column=='center'):
                keyboard.press_and_release('left')
                # print(column)
            if(column=='left'):
                keyboard.press_and_release('left')
                keyboard.press_and_release('left')
                # print(column)
        if(column_copy=='center'):
            if(column=='left'):
                keyboard.press_and_release('left')
                # print(column)
            elif(column=='right'):
                keyboard.press_and_release('right')
                # print(column)
        elif(column_copy=='left'):
            if(column=='center'):
                keyboard.press_and_release('right')
                # print(column)
            elif(column=='right'):
                keyboard.press_and_release('right')
                keyboard.press_and_release('right')
                # print(column)
    if(fwd_bwd_copy!=fwd_bwd and fwd_bwd=='Backward'):
        # print('jump')
        keyboard.press('space')
```

change `keyboard.press_and_release()` with the key controls of your game.Here `fwd_bwd` works for jump(left hand) and `str_no_str` is for right hand controls.

5 Experiments and results

5.1 Keyboard Presses v/s Socket

Keyboard presses are extremely slow and lag very often. Socket on the other hand is very clumsy to implement. We need to run the client file and then we need to run the PoseNet and then finally we can start the game which of course takes input from the client file. Hence we implemented keypresses. But the problem with keypresses is that it keeps on pressing the keys and interferes with your work. Sometimes it becomes difficult to control the applications just because it keeps on pressing the keys.

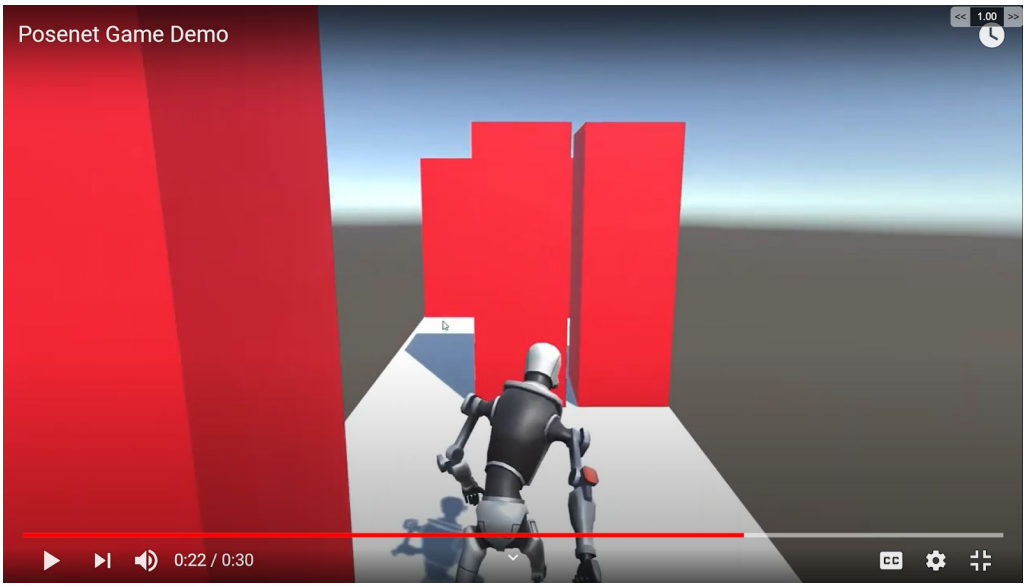
In Future, we plan to implement a better strategy.

5.2 Response Time

As said earlier, the response times are slower and the game gets out of control. Socket has a relatively quick response. For the next task , all we should do is make a game controller library which does not use any intermediary means, i.e. in-game implementation.

5.3. Demo link

click on the image



6. CONCLUSION AND FUTURE WORK

6.1.Conclusion

To sum up, ConvNet Architectures are amazing and Computer Vision is the best approach rather than algorithmic approaches. End-to-End deep learning is the future, considering huge amounts of data available.

Pose Estimation could be better with a better architecture.PoseNet does not work fast enough on today's usual computing machines. We could overcome this difficulty by implementing better architectures and faster decoding algorithms.One other way we can achieve it is using C++ instead of python.

Openpose uses C++ and it goes upto 8 FPS on a machine with GPU NVidia GTX 1650 with around 850 CUDA cores, whereas PoseNet-Python reaches 2 - 4 FPS. Openpose uses OpenGL for rendering the overlaying image whereas PoeNet uses OpenCV. Openpose detects upto 70 Facial keypoints, 6 foot keypoints, 2x21 hand keypoints. These differences have gaurenteed the scope for further improvement.

Also, detecting 3D coordinates is a challenge.

6.2 3D Pose

In future, we aim to have our own model for 3D Pose Estimation. For that we need to customize the loss functions according to the depth of the monocular rgb image. With that said, we need to devise a depth function, which would be matched with the depth in Ground Truth (The Raw mocap data), thus generating predictions that match with it. This is an extremely difficult task.

6.3 C++ Port

Porting is a big task. It's favourable because of the fact that C++ is a lot faster than Python. A millisecond is forever in today's world. It could be avoided altogether if we could train and implement it all over from the beginning in C++.

6.4 Portability

By portability we mean that it could be released as a Python Package / C++ library which could be easily used for a wide variety of applications. One of them being Avitra.

7.References

[0] https://github.com/tensorflow/tfjs-models/tree/master/posenet

[1] https://github.com/CMU-Perceptual-Computing-Lab/openpose

[2] https://github.com/xingyizhou/Pyt

[4] https://arxiv.org/pdf/1603.06937.pdf

[5] https://arxiv.org/pdf/1701.01779.pdf

[6] https://arxiv.org/pdf/1803.08225.pdf

[7] https://github.com/leerob/Space_Invaders

[8]https://www.youtube.com/watch?v=6R1GdS68p7A&feature=youtu.be

[9]https://github.com/ildoonet/tf-pose-estimation/tree/6980660b6f50653646a33c5a493d4c51d4335a3f/src

Related :

https://github.com/cbsudux/awesome-human-pose-estimation

https://github.com/rwightman/posenet-python

https://nanonets.com/blog/human-pose-estimation-3d-guide/amp/#monocular-deep-cnn

https://github.com/dronefreak/human-action-classification

https://arxiv.org/pdf/1612.06524.pdf