计算机2104 高一辰 20215921

# 三维人体姿态的捕捉与动画实现

## 一、背景介绍

动作捕捉，通常是指在3D游戏或动画中，通过传感器和软件，把真人演员的动作转录成数字模型的动作。众所周知，动画和游戏中的角色（包括人物和动物）必须要有动作，比如奔跑、跳跃、打斗等等。很久以前，这些动作都是需要人工一帧一帧画上去的。后来，又有了先进一点的动作捕捉技术，系统里有一套“电子戏服”，衣服上的电位器通过数据线和电脑相连接。真人演员穿上戏服表演，演出动作会被实时传输到电脑，转换成相应的关键帧数据。这样，原本需要手K好几天的动作，用动捕系统在几分钟内就能完成了。

但是，一套动捕设备的价格不菲，市面上较为常用的就是惯性动作捕捉系统、光学动作捕捉系统，价格从几万到十几万都有，不仅个人开发者用不起，甚至对小团队来说也是很大的压力。因此，便宜大众而效果好的视频动作捕捉技术有很大的市场潜力。2019年起，苹果推出AR kit 3.0，可以通过单目摄像头捕捉人物动作，优点就是用起来方便+便宜，但是效果不好，稳定性差。因此，这个领域的技术还需要打磨。希望在不久的未来，做动画会和拍抖音短视频一样容易。

## 二、问题分析

### （1）任务

对于给定的人物（单人或多人）视频，利用模型提取人体关键关节点，分割不同的人体，最后把同样的动作施加到到unity模型上。

### （2）难点

1. 每张图片中包含的人的数量是未知的。

2. 人与人之间的相互作用是非常复杂的，比如接触、遮挡等，这使得联合各个肢体，即确定一个人有哪些部分变得困难。

### （3）技术方案及步骤

1.先进行2D 姿态估计：如图一所示，{1-颈部, 2-右肩, 3-右肘, 4-右手腕, 5-左肩, 6-左肘, 7-左手腕, 8-右髋部, 9-右膝, 10-右踝, 11-左髋部, 12-左膝, 13-左踝, 14-左眼, 15-右眼, 16-左耳, 17-右耳}，只需指定这17 个关键点（landmark points）即可完整描述人体姿势。

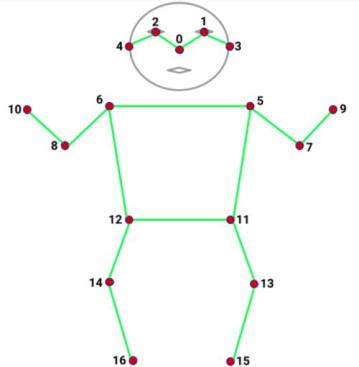


图 1 人体关键点示意图

2.利用ALphapose框架来检测视频中人类的姿势，并跟踪他们的动作，以便能够构建平滑的移动动画。

1. 上一步获得的坐标是 2D 的，现在需要将这些 2 维坐标映射到 3D 空间中，采用的办法是3d-pose-baseline。

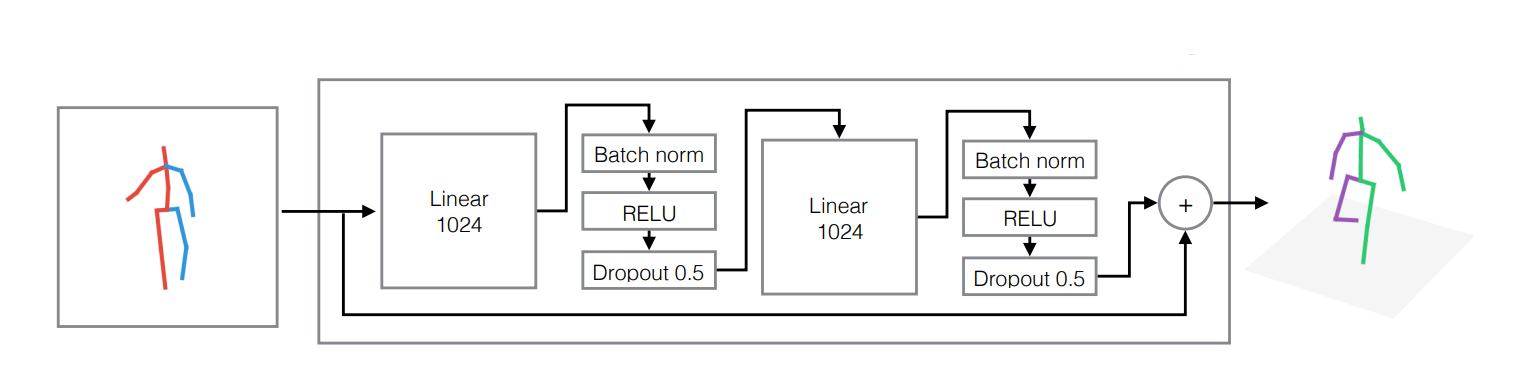


图 2 原理图

1. 从姿势估计框架中获得了关键点的三维坐标后，就可以将这些坐标提供给 3D 角色身体的每个肢体。使用Unity作为3D动画环境来完成任务，使用开源的 unity model。每帧的 17 个关键点的坐标存储在文本文件中，并在 Unity 中使用 C# 读取的。然后，把从文件中读取的坐标重新定位到 3D 人形模型中，与人形头像的身体关键点进行映射。动画是使用 Unity 的逆运动学（inverse kinematics）、骨骼旋转和四元数完成的。

## 三、数据集

| Dataset name | Download command |
| --- | --- |
| COCO 2017 | mim download mmpose --dataset coco2017 |
| MPII | mim download mmpose --dataset mpii |

COCO网址：https://cocodataset.org/#download

MPII：http://human-pose.mpi-inf.mpg.de/

## 四、实验分析

### （1）开发环境

Windows11

NVIDIA GeForce RTX 4060

Annaconda

Unity

### （2）实验设置、数据、以及对比结果

如下图所示：



图 3 待分割图像



图 4 计算分割框

第123帧的输出的数据含义分别为：表示该图像帧中检测到的框的编号、第一个框的左上角 x 坐标、左上角 y 坐标、右下角 x 坐标、右下角 y 坐标、第二个框的标签、左上角 x 坐标、左上角 y 坐标、右下角 x 坐标、右下角 y 坐标。

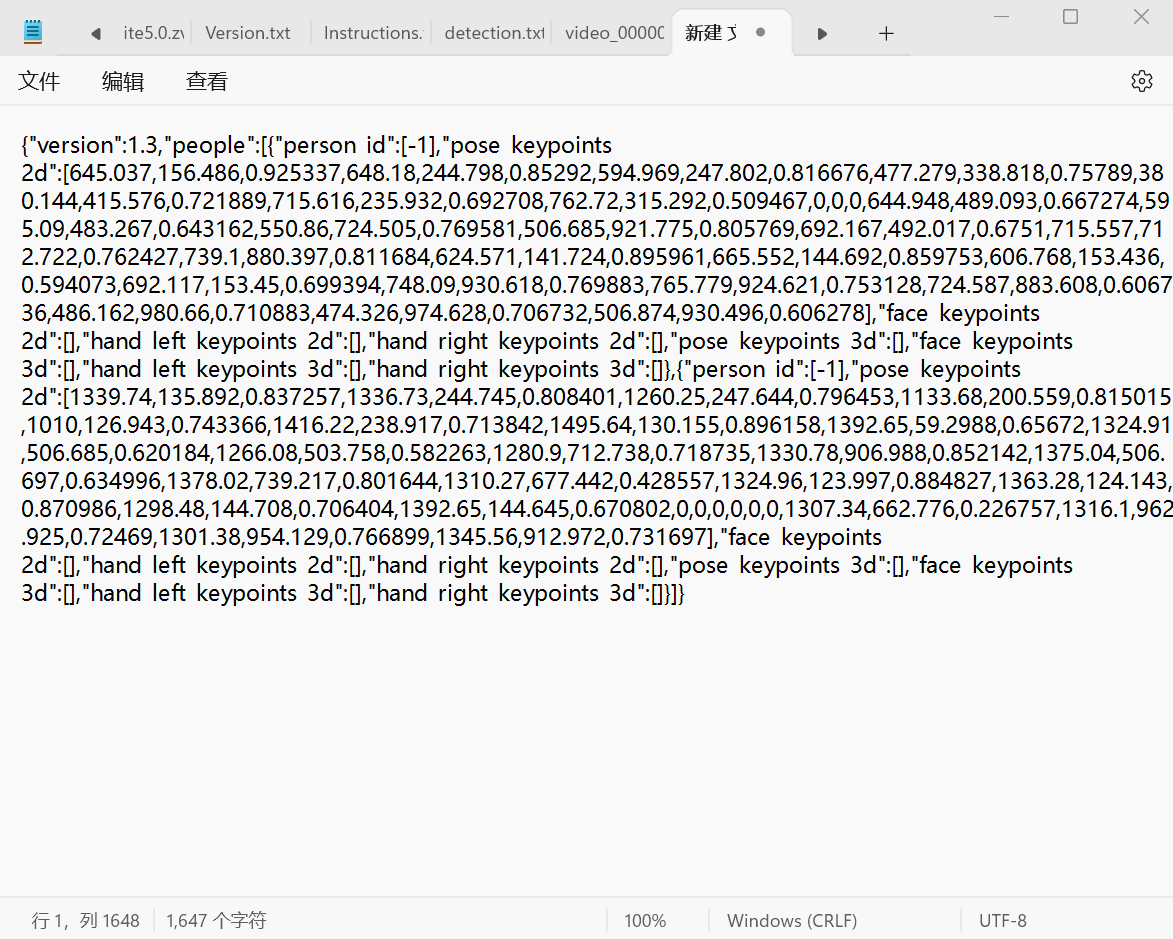


图 5 json文件输出一帧的关键点信息

pose\_keypoints\_2d是一个包含关节坐标和置信度的列表，共有17个关节。例如，第一个关节的坐标为(645.037, 156.486)，置信度为0.925337。关节的顺序遵循人体姿势估计的标准。



图 6 导出视频效果

### 对比实验

Results on COCO test-dev :

| Method | AP @0.5:0.95 | AP @0.5 | AP @0.75 | AP medium | AP large |
| --- | --- | --- | --- | --- | --- |
| OpenPose (CMU-Pose) | 61.8 | 84.9 | 67.5 | 57.1 | 68.2 |
| Detectron (Mask R-CNN) | 67.0 | 88.0 | 73.1 | 62.2 | 75.6 |
| AlphaPose | 73.3 | 89.2 | 79.1 | 69.0 | 78.6 |

Results on MPII full test set:

| Method | Head | Shoulder | Elbow | Wrist | Hip | Knee | Ankle | Ave |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| OpenPose (CMU-Pose) | 91.2 | 87.6 | 77.7 | 66.8 | 75.4 | 68.9 | 61.7 | 75.6 |
| Newell & Deng | 92.1 | 89.3 | 78.9 | 69.8 | 76.2 | 71.6 | 64.7 | 77.5 |
| AlphaPose | 91.3 | 90.5 | 84.0 | 76.4 | 80.3 | 79.9 | 72.4 | 82.1 |

### 现有缺点

特殊场景下检测效果差。比如人体姿势比较诡异的时候，或者存在遮挡时，模型会有错位现象。此外，模型的舞蹈动作没有真人视频流畅，想要达到比较好的效果，还需要进一步的逐帧调整。

## 五、完整的程序源代码

【1】处理视频文件，生成逐帧的jpg

import cv2 # 导入OpenCV库（用于处理视频和图像）

import os # 导入os库

# 关于按帧抽取图片的函数

def FrameCapture(path): # 定义一个名为FrameCapture的函数，参数是视频的路径

vidObj = cv2.VideoCapture(path) # 创建一个VideoCapture对象，打开指定路径的视频文件。

count = 0 # 计数器，记录提取的帧数。

success = 1 # 标志，检查是否成功提取帧。

while success: # 循环，直到无法继续提取帧为止。

success, image = vidObj.read() # 调用read函数从视频中读取一帧图像，并将结果赋值给success和image变量。success表示是否成功读取帧，image是读取到的图像。

cv2.imwrite("./frames/frame%d.jpg" % count, image) # 将读取到的帧保存为图像文件，文件名以帧计数命名，保存在"./frames/"文件夹中。

count += 1 # 计数器自增，记录已经提取的帧数。

if \_\_name\_\_ == '\_\_main\_\_': # 判断当前模块是否作为主程序运行。

FrameCapture("./multi\_video.mp4") # 调用FrameCapture函数，传入视频文件路径"./multi\_video.mp4"，开始提取视频帧并保存到指定路径。

【2】

%matplotlib inline

import matplotlib.pyplot as plt

import matplotlib.patches as patches

from PIL import Image

import numpy as np

#导入库

im = np.array(Image.open('frame122.jpg'), dtype=np.uint8)

# 创建图像和坐标轴

fig, ax = plt.subplots(1)

# 显示图像

ax.imshow(im)

# 创建矩形补丁

rect = patches.Rectangle((994.7803417731519, 270.29939404752565), 265, 566, linewidth=1, edgecolor='r', facecolor='none')

rect2 = patches.Rectangle((510.72019953723805, 252.42652621274465), 180.04, 546.488, linewidth=1, edgecolor='r', facecolor='none')

# 将补丁添加到坐标轴

ax.add\_patch(rect)

ax.add\_patch(rect2)

plt.show()

【3】

file1 = open("./detection.txt", "r+") # 打开文件"./detection.txt"，读写模式打开。

line1 = file1.readline() # 读取文件的第一行内容。

# print(line1)

all\_boxes = [] # 创建空列表，用于存储所有的框坐标。

while line1: # 循环读取每一行的内容。

line1 = line1.split(" ") # 将读取到的行内容按空格进行分割，得到一个字符串列表。

new\_list = [] # 创建一个空列表，用于存储每个框的坐标信息。

if len(line1) == 2: # 如果字符串列表长度为2，说明是只包含一个整数的行（表示框的数量）。

new\_list.append(int(line1[0])) # 将字符串转换为整数，并添加到new\_list中。

all\_boxes.append(new\_list) # 将new\_list添加到all\_boxes列表中。

if len(line1) != 2: # 如果字符串列表长度不为2，说明是包含框坐标信息的行。

for i in range(len(line1) - 1): # 遍历字符串列表中除最后一个元素以外的元素。

if i == 0: # 如果是第一个元素，表示框的标签。

new\_list.append(int(line1[i])) # 将字符串转换为整数，并添加到new\_list中。

elif (i - 1) % 5 == 0: # 如果是第二个及以后每隔5个元素，表示框的坐标。

new\_list.append(int(line1[i])) # 将字符串转换为整数，并添加到new\_list中。

else: # 其他情况为浮点数，表示框的坐标值。

new\_list.append(float(line1[i])) # 将字符串转换为浮点数，并添加到new\_list中。

all\_boxes.append(new\_list) # 将new\_list添加到all\_boxes列表中。

line1 = file1.readline() # 读取文件的下一行内容。

file1.close() # 关闭文件。

【4】在每一帧中绘制人体周围的边界框

count = 0

for i in range(len(all\_boxes)):

if len(all\_boxes[i]) != 1:

print(all\_boxes[i])

# 读取图像

im = np.array(Image.open('./frames/frame' + str(all\_boxes[i][0]) + '.jpg'), dtype=np.uint8)

# 创建图像和坐标轴

fig, ax = plt.subplots(1)

# 显示图像

ax.imshow(im)

# 创建矩形框1

rect = patches.Rectangle((all\_boxes[i][2], all\_boxes[i][3]), all\_boxes[i][4] - all\_boxes[i][2], all\_boxes[i][5] - all\_boxes[i][3], linewidth=1, edgecolor='r', facecolor='none')

# 创建矩形框2

rect2 = patches.Rectangle((all\_boxes[i][7], all\_boxes[i][8]), all\_boxes[i][9] - all\_boxes[i][7], all\_boxes[i][10] - all\_boxes[i][8], linewidth=1, edgecolor='r', facecolor='none')

# 将矩形框添加到坐标轴

ax.add\_patch(rect)

ax.add\_patch(rect2)

plt.show()

count += 1

if count == 40:

Break

【5】

print(len(all\_boxes)) # 打印all\_boxes列表的长度

list\_of\_files = {} # 创建一个空字典，用于存储文件名和文件路径的映射关系

path = ".\\output" # 指定文件夹路径

for (dirpath, dirnames, filenames) in os.walk(path):

for filename in filenames:

if filename.endswith('.json'):

# 将文件名和文件路径添加到字典中

list\_of\_files[filename] = os.sep.join([dirpath, filename])

【6】检查姿势关键点是否在边界框内的函数

def FindPoint(x1, y1, x2, y2, x, y):

# 判断点(x, y)是否在矩形区域(x1, y1, x2, y2)内

if (x > x1 and x < x2 and y > y1 and y < y2):

return True

else:

return False

import json

count = 0

main\_thing = 0

# 打印list\_of\_files字典中的键的数量

print(len(list\_of\_files.keys()))

# 遍历list\_of\_files字典的键

for i in list\_of\_files.keys():

# 获取文件路径

list\_of\_files[i]

# 打开JSON文件并加载数据

with open(list\_of\_files[i]) as json\_file:

data = json.load(json\_file)

# 如果people列表为空或all\_boxes中的元素个数为1，则将数据保存到两个文件中

if len(data["people"]) == 0 or len(all\_boxes[count]) == 1:

with open('.\\persons\\1\\1' + i, 'w') as outfile:

json.dump(data, outfile)

with open('.\\persons\\2\\2' + i, 'w') as outfile:

json.dump(data, outfile)

# 如果people列表不为空且all\_boxes中的元素个数大于1

if len(data["people"]) != 0 and len(all\_boxes[count]) > 1:

for person in data["people"]:

key\_point\_count1 = 0

key\_point\_count2 = 0

# 遍历人物的关键点列表

for keypoints in range(0, len(person["pose\_keypoints\_2d"]), 3):

x = person["pose\_keypoints\_2d"][keypoints]

y = person["pose\_keypoints\_2d"][keypoints + 1]

print(count)

# 判断关键点是否在all\_boxes[count]表示的矩形区域内

k = FindPoint(all\_boxes[count][2], all\_boxes[count][3], all\_boxes[count][4], all\_boxes[count][5], x, y)

# 根据判断结果和all\_boxes[count]中的第一个元素的值更新关键点计数器

if k == True and all\_boxes[count][1] == 1:

key\_point\_count1 += 1

elif k == True and all\_boxes[count][1] == 2:

key\_point\_count2 += 1

try:

# 如果all\_boxes[count]中的元素个数大于6，则处理另外两个关键点

k = FindPoint(all\_boxes[count][7], all\_boxes[count][8], all\_boxes[count][9], all\_boxes[count][10], x, y)

if k == True and all\_boxes[count][6] == 1:

key\_point\_count1 += 1

elif k == True and all\_boxes[count][6] == 2:

key\_point\_count2 += 1

except:

print("oh\_boy")

# 根据关键点计数器的值将人物数据保存到不同的文件中

if(key\_point\_count2 > key\_point\_count1):

print("person2")

with open('.\\persons\\2\\2' + i, 'w') as outfile:

data1 = {"version": 1.3}

data["people"] = []

data["people"].append(person)

json.dump(data, outfile)

else:

print("person1")

with open('.\\persons\\1\\1' + i, 'w') as outfile:

data1 = {"version": 1.3}

data["people"] = []

data["people"].append(person)

json.dump(data, outfile)

print("ok check")

count += 1

【7】charanim.cs 的渲染更新函数 update()

(charanim.cs 中的代码使用四元数来制作骨骼动画)

void Update()

{

if (pos == null) {

return; // 如果pos为空，退出函数

}

play\_time += Time.deltaTime; // 计算播放时间

int frame = s\_frame + (int)(play\_time \* 24.057f); // 计算当前帧数

if (frame > e\_frame) {

play\_time = 0; // 如果超过结束帧数，重置播放时间和帧数

frame = s\_frame;

}

if (debug\_cube) {

UpdateCube(frame); // 更新Cube（立方体）的位置和旋转

}

Vector3[] now\_pos = pos[frame]; // 获取当前帧的位置信息

Vector3 pos\_forward = TriangleNormal(now\_pos[7], now\_pos[4], now\_pos[1]); // 计算方向向量

bone\_t[0].position = now\_pos[0] \* scale\_ratio + new Vector3(init\_position.x, heal\_position, init\_position.z); // 更新骨骼位置

bone\_t[0].rotation = Quaternion.LookRotation(pos\_forward) \* init\_inv[0] \* init\_rot[0]; // 更新骨骼旋转

for (int i = 0; i < bones.Length; i++) {

int b = bones[i];

int cb = child\_bones[i];

bone\_t[b].rotation = Quaternion.LookRotation(now\_pos[b] - now\_pos[cb], pos\_forward) \* init\_inv[b] \* init\_rot[b];

// 更新骨骼旋转，通过LookRotation函数计算旋转角度

}

bone\_t[8].rotation = Quaternion.AngleAxis(head\_angle, bone\_t[11].position - bone\_t[14].position) \* bone\_t[8].rotation;

// 更新骨骼旋转，通过AngleAxis函数计算旋转角度

}

**其中的 updateCube() 定义如下：**

void UpdateCube(int frame)

{

if (cube\_t == null) {

cube\_t = new Transform[bone\_num]; // 创建一个Transform数组用于存储Cube的引用

for (int i = 0; i < bone\_num; i++) {

Transform t = GameObject.CreatePrimitive(PrimitiveType.Cube).transform; // 创建一个Cube对象

t.transform.parent = this.transform; // 设置Cube对象的父对象为当前对象

t.localPosition = pos[frame][i] \* scale\_ratio; // 设置Cube对象的本地位置为给定帧上的位置信息

t.name = i.ToString(); // 设置Cube对象的名称为当前索引值

t.localScale = new Vector3(0.05f, 0.05f, 0.05f); // 设置Cube对象的缩放比例

cube\_t[i] = t; // 将Cube对象的引用存储到数组中

Destroy(t.GetComponent<BoxCollider>()); // 移除Cube对象的BoxCollider组件

}

}

else {

Vector3 offset = new Vector3(1.2f, 0, 0); // 创建一个偏移量向量

for (int i = 0; i < bone\_num; i++) {

cube\_t[i].localPosition = pos[frame][i] \* scale\_ratio + new Vector3(0, heal\_position, 0) + offset;

// 设置Cube对象的本地位置为给定帧上的位置信息乘以缩放比例，并添加偏移量

}

}

}

## 六、引用文献

[1] Cao Z., Simon T., Wei S., Sheikh Y. Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 1302-1310.

[2] Wei S., Ramakrishna V., Kanade T., Sheikh Y. Convolutional Pose Machines. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 4724-4732.

[3] Bewley A., Ge Z., Ott L., Ramos F., Upcroft B. Simple Online and Realtime Tracking with a Deep Association Metric. In: Proceedings of the IEEE International Conference on Image Processing (ICIP), 2016, pp. 3645-3649.

[4] Zhou X., Andriluka M., Olafsson S., Leonardis A., Schiele B. 3D Human Pose Estimation: A Review of the Literature and Analysis of Covariates. In: Computer Vision and Image Understanding, 2019, vol. 179, pp. 102-123.

[5]Martinez, J., Hossain, R., Romero, J., & Little, J. J. (2017). A simple yet effective baseline for 3D human pose estimation. In Proceedings of the IEEE International Conference on Computer Vision (ICCV).