3D Parkour Game Manipulated with Real-Time Pose Estimation Technique

111061516 Jin-Yu Young, 111061564 Hao-Jiun Tu, 111061585 Jing-Chun Wang
111061519 Mei-Yen Tsai, 107081028 An-Yan Chang

Abstract

Human pose estimation is an important task in computer vision to identify the position of a human body in a specific scene. In this project, focusing on the application, we utilize the techniques, i.e., 3D human pose estimation in our Parkour game. With the state-of-the-art work, VNect [4], which supports real-time 3D pose estimation, has an accurate prediction on human joints. We further design the Parkour game and use the predicting information to Unity platform to fulfill the game like Xbox 360 - Kinect Sports (Fig. 1). Finally, our game has been successfully completed with 23 frames per second on average (nearly real-time specification), and we also record a demo video to demonstrate our project achievement.

1. Introduction

1.1. Human Pose Estimation in Video

Human pose estimation is a task in computer vision that focuses on identifying the position of a human body in a specific scene. The essence of the technology lies in detecting points of interest on the limbs, joints, and even faces of a human. These key points are used to produce a 2D or 3D representation of a human body model. The model must identify the fine-grained joint coordinates for the human body. For a multi-frame human pose estimation, it takes a large number of resources, such as memory bandwidth and computing power, to have a great performance in complicated scenes. It is a tradeoff between the computing resource and the accuracy. In this project, we want to improve this technology, i.e., human pose estimation and video person pose tracking in real-time constraints.

1.2. Application

In the project, we want to combine the aforementioned technology with some video games like Xbox 360 - Kinect Sports in Fig. 1. There is a camera sensor that will capture the human body of players, and the action and movement of characters in the game will be reflected by players. Parkour game seems to be an appropriate application for this technology. It requires real-time feedback and is suitable for body manipulation. Furthermore, parkouring as a personal experience would be such entertainment. Therefore, we want to design a parkour game with the character controlled by the players with their poses. The conceptual graph of the Parkour game we expected is shown in Fig. 2.

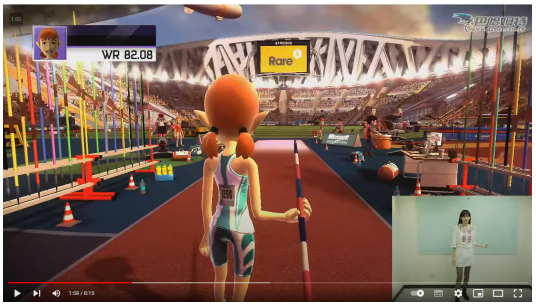

Figure 1. Xbox 360 - Kinect Sports.

Figure 2. Conceptual graph of Parkour game.

2. Related Work

2.1. Recap of Previous Work

Pose estimation is the task of determining the position and orientation of an object or a person in an image or a video. It is a fundamental problem in computer vision and has a wide range of applications, including augmented reality, robotics, and human-computer interaction.

In the field of human pose estimation, there have been various methods proposed over the years. Early approaches used hand-crafted features and shallow learning architec-

tures, such as pictorial structures and conditional random fields. These methods were soon replaced by deep learning-based approaches that leverage convolutional neural networks (CNNs) to learn discriminative features from images.

One of the most widely used CNN architectures for human pose estimation is the convolutional pose machine (CPM), proposed by Wei et al [7] in 2016. CPM uses a CNN to extract features from an input image and applies a series of convolutional and max-pooling layers to generate heatmaps for each joint.

Recent advancements have led to more sophisticated methods and architectures that can improve the estimation accuracy. For example, some studies have used multi-person pose estimation architectures that use Part Affinity Field (PAF) representation, such as the OpenPose, proposed by Cao et al [2], which is to learn to associate body parts with individuals in the image. This bottom-up system achieves high accuracy and real-time performance.

In object pose estimation, the dominant approach is using CNNs with PoseNet [3], as a heatmap-based method, that can estimate the position of an object by regressing the 3D coordinates of its key points. Another common method is using single-shot detection with pose estimation, such as YOLOv3 [5] and RetinaNet [6].

In conclusion, there is an active research in the field of pose estimation, with various techniques and architectures proposed to improve the accuracy and efficiency of the task. the research trend shifting to real-time and multi-person scenarios.

2.2. 3D Human Pose Estimation

The state-of-the-art work, VNect model [4], will be our first choice to realize his technology, i.e., 3D human pose estimation in video. Fig. 4. provides an overview of VNect method to fulfill real-time 3D human pose estimation. It consists of two primary components. The first is a CNN model to regress 2D and 3D joint positions. The second component combines the regressed joint positions with a kinematic skeleton fitting method to produce a temporally stable, camera-relative, full 3D skeletal pose. Fig. 6. shows the model architecture of VNect, which basically adopts the backbone from ResNet50. It predicts the 2D heatmap H and 3D location-maps X, Y, Z to form the kinematic skeleton. Further, we will use the information as our Unity input to design the Parkour game. Our main contribution in the project is to utilize the pose estimation techniques in the game, which would encounter many difficulties such as the format mismatch for the model and Unity platform, model output unmatching with the character in the game, and the design of the scene in the game. The details will be introduced in the following sections.

3. Technical Part

3.1. Overview

Fig. 3 shows our workflow in this project. First, we utilize VNect as our backbone model to generate the correct bounding box, and use the information to produce heatmaps and location maps (offset maps), which can precisely locate the position of each human joint. To implement neural network models in Unity framework, we use Barracuda package, which is a lightweight cross-platform neural network inference library for Unity. We will store the well-trained model using PyTorch or TensorFlow in ONNX format. ONNX is an open file format designed for machine learning. We will use Unity Barracuda to import pretrained Vnect models, allowing the model to perform real-time inference in Unity.

3.2. Implementation Details

3.2.1 Prediction Methods

Fig. 4 shows the overview of prediction methods. Roughly, the 3D pose estimation can be divided into two parts: bounding box tracking and heatmaps/location maps (offset maps) predictions. For the bounding box tracking, to guarantee the real-time constraint, we have to restrict the size of the input to the network and track the scale of the person in the image to avoid searching the scale space in each frame. The approach here is that 2D pose predictions at each frame are used to determine the bounding box for the next frame. Once the bounding box is decided, the image will be resized to 368x368 and sent into the CNN model to generate heatmaps and offset maps. Basically, input images for CNN regression are divided into 28 blocks, and each of human joint corresponds to one of the 28 blocks. The heatmap represents the probability for each human joint in 28 blocks, and the offset map stands for the offset of coordinates in the center of a block. Fig. 5 shows the 2D schematic diagram of the heatmap and offset map. In the implementation, the region with the maximum probability of a heatmap can coarsely decide the joint position, and we can further get the accurate joint position via the calibration of the offset map.

3.2.2 Model Structure

In this project, we utilize Vnect to conduct bounding box tracking and extract heatmaps and offset maps, since it can support real-time 3D pose estimation. The model architecture is shown in Fig. 6. ResNet50 network as a backbone, the layers of ResNet50 from res5a onwards are replaced with the architecture depicted in Fig. 6.

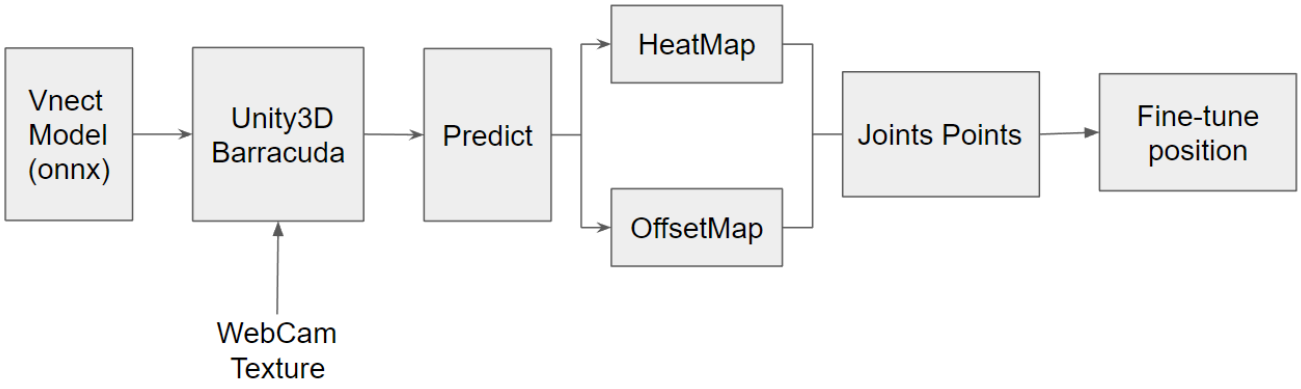

Figure 3. Our workflow in this project.

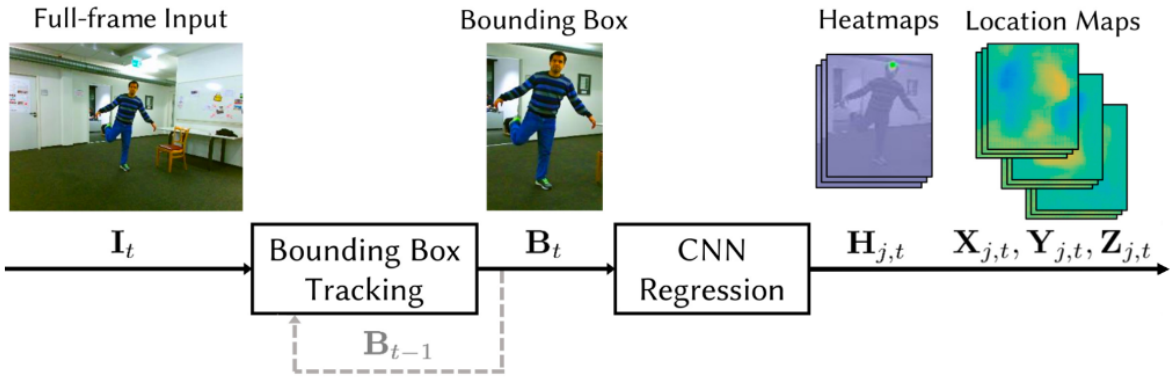

Figure 4. The overview of prediction methods.

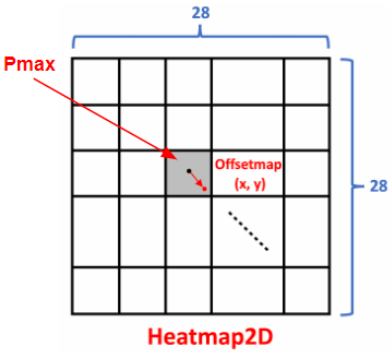

Figure 5. 2D example for the heatmap and offset maps.

3.2.3 Fine Tuning - Output Unmatch with Character

We choose "Unity Chan", which is released officially by Unity for free, as our main character in the game. She is built meticulously with 153 joint points, composed of organs and clothing as Fig. 7 shown. Since the body structure of Unity Chan is too complete (153 joint points), while our model only supports 24 joint points, the coordinates of most of her body parts are calculated. To solve the problem, we have to simplify excessive joint points. Some of our estimations exhibit as below:

- Hip: Locate hip position as the middle of the roots of left and right thighs. Adjust the z-positional offset with the bottom of her spine.
- Neck: Locate the neck position with both shoulders.
- Head: Locate the head parental position with the top of her head and the position of her neck.
- Facial feature: The positions follow the head position which follows the neck position.
- Spine: Locate the spine parental position with her upper abdomen.
- Fingers and toes: The position of fingers and toes follow their parent components.

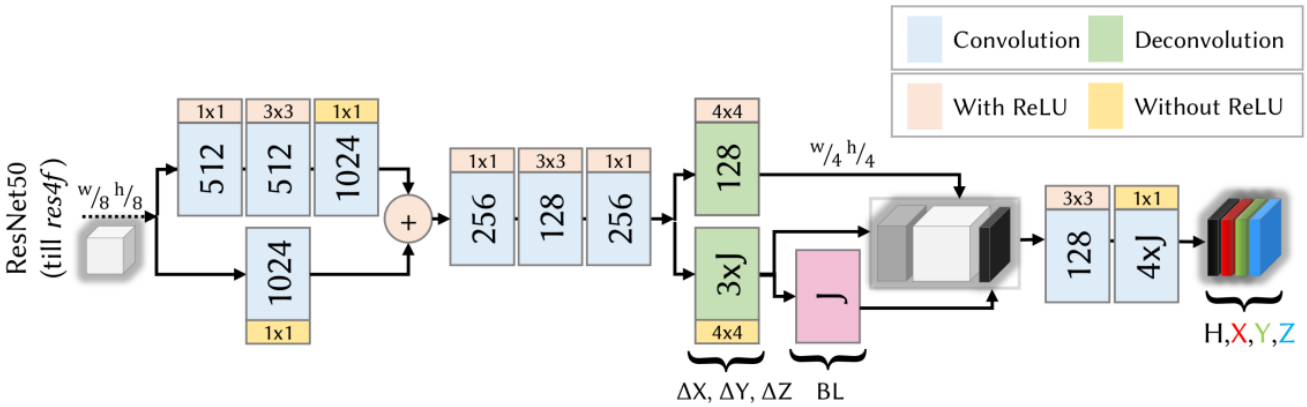

Figure 6. Network structure for VNect. The structure above is preceded by ResNet50 till level 4. The network predicts 2D location heatmaps H and root relative 3D joint locations X, Y, Z.

- Clothing: Her upper clothing follows her shoulders and spine, while her lower clothing follows her spine, hip, and legs.

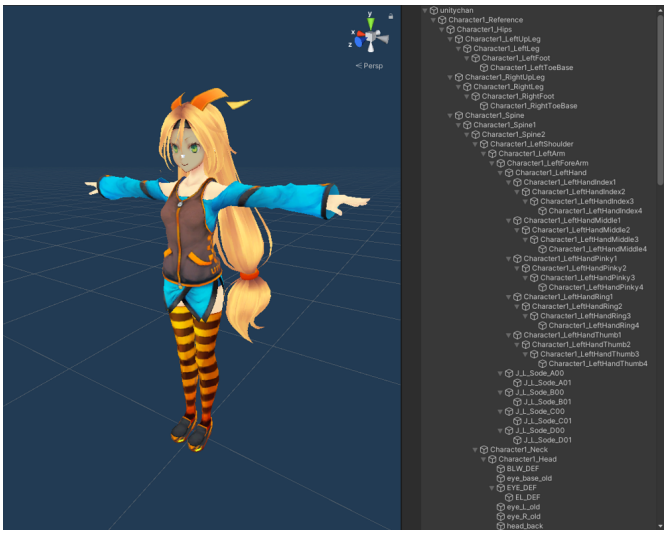

Figure 7. Unity Chan and part of her components

3.2.4 Fine Tuning - Demonstrating in Game

Since a parkour game requires wide space to show players' movement, our game is designed with a large scene. However,the webcam is not able to capture a huge enough area. We have to scale up the offset when the character shifts horizontally. Based on the shifting offset of the hip position from the previous position, we adjust the multiplication until it looks suitable in gaming as shown in Fig. 8. Apart from that, due to the skeleton mechanism in Unity, Unity Chan could traverse her whole body according to the shifting of her hip. In this case, her body wouldn't be folded or unnaturally stretched.

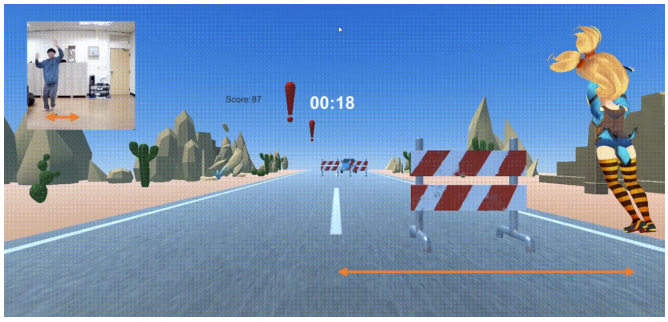

Figure 8. Scaling the shifting offset in Unity

Although the coordinates of each organ are set perfectly, the rotation should be appropriate to look natural. Here, we use the "LookRotation" function to specify the forward and upward direction. Some of our implementations are shown below:

- Hip: Forward direction is the triangle normalization of hip position, left and right thigh bending position.
- Gaze: Gazing direction was set as the vector from head position to nose position.
- Hands: Forward direction was the vector from the thumb to the middle finger position. The upward direction was the triangle normalization hand position, middle finger position, thumb position. With these looking rotation adjustments, Unity Chan could move naturally with no abnormal distortion.

3.2.5 Unity Game Design

In the game, we need an infinite runway, which is made as in Fig. 9. Assuming that we make scenes A, B, and C, the size of one scene is the camera's field of view. By connecting the scenes together, we need to set three parameters: speed, bound, and offset. The connected scenes must move in the same direction as the camera, creating the effect of moving forward. The speed parameter controls the overall movement speed of the connected scenes. If the current scene within the camera's field of view is A, and the boundary of A reaches the set bound position, it means that scene A has gone out of the camera's field of view. At this time, scene A will move backward by an offset length, making it connect to the last scene C. By controlling these three parameters, we can achieve infinite scrolling background.

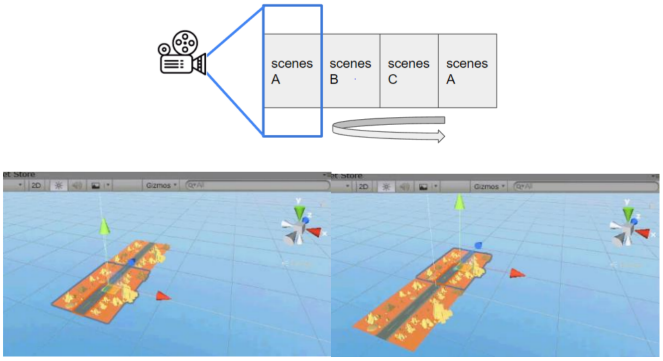

Figure 9. Infinite Scrolling Background

As we have two types of scoring items shown in Fig. 10, one being obstacles on the ground that subtract points, and the other being items in the sky at a certain height that adds points, for the convenience of project management, here we will split them into two items: add points and subtract points, creating sub-items within the add points and subtract points items, so that the sub-items can be adjusted on the y-axis (height) uniformly within the same item.

For example, in the subtract points item, an obstacle will be generated at a given time, we need to set what items may appear, and on the ground, we need to set all the possible positions for the items. After setting the items and positions, we need to handle the problem of random generation, which can be split into two parts, random generation of positions and random generation of items. At a given time, we will randomly select a position for the obstacle to be generated from the positions that have been set, and randomly select the item that will be generated from the items that have been set.

The time interval between generations is adjusted according to the background moving speed, assuming the background moving speed is 10 meters per second, and the depth of a single scene A is 150 meters, it means that the time interval of the next scene's obstacle generation is 15 seconds. Currently, we have set 5 positions on the ground for generating items, these 5 positions will be randomly generated in sequence. Therefore, in 15 seconds all five positions will be generated in sequence. The time interval between each item's generation will be at most 3 seconds, here we set it to 2.5 seconds, to ensure that when the camera reaches the scene, all obstacles have been completely generated, so that there will be enough reaction time for the players.

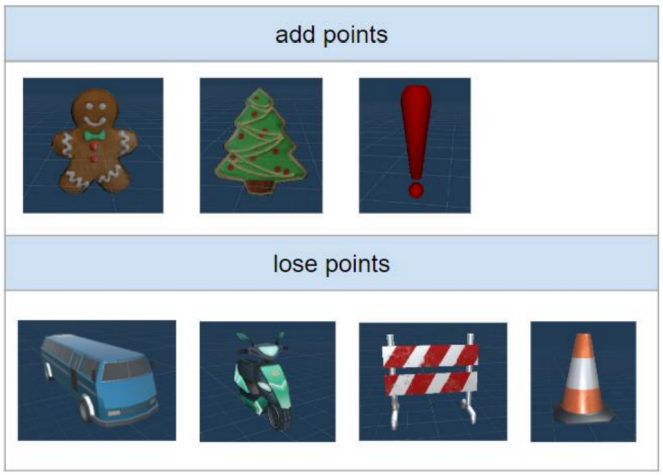

Figure 10. Obstacles in game

Regarding collision condition settings, we split it into add points and subtract points settings. Using the subtract points as an example, we will set the collision area for the player's game character and the obstacle on the screen, and one of them must be set to a trigger, so that when the character and obstacle come into contact, it will trigger a certain event. This event is to control the score subtraction.

In order to increase the game's difficulty, we gradually increase the speed of the screen and at the same time, we must also adjust the random obstacle generation time accordingly. The background speed starts at 10 meters per second, and every 2.5 seconds the speed is increased by 1. Here the background speed growth rate is linear. To ensure that all obstacles are generated completely before the camera reaches the next scene, we use an exponential decay for the obstacle generation interval time.

4. Experiments

Finally, our Parkour game has been successfully completed, including the model prediction transferring to Unity platform scene arrangement, obstacles that add and subtract points, and so forth. The interface of Unity in our game is displayed in Fig. 11. We also have filmed a simple demo video [1] to demonstrate our project achievement.

Figure 11. Our Parkour game in the interface of Unity.

At present, 3D human pose estimation utilized in our Parkour game can detect any big movement and finest action well, and it is also smooth while playing the Parkour game. After the testing of timing analysis function in Unity, our Parkour game achieves 23 frame per second in average, which is nearly the real-time specification in electronic devices.

5. Conclusions

In the project, we implemented an entertaining Parkour game with 3D human pose estimation techniques. By using the state-of-the-art network, VNect, which supports real-time pose estimation application, the connection between model output and Unity interface, and the game design in Unity, our Parkour game achieves 23 frames per second on average, which is nearly the real-time specification in electronic devices and reaches our project goal set in the beginning. In future work, we can survey another advanced model which is compact and fast, or some model compression techniques which don't degrade the performance too much, to further improve the smoothness while playing games.

References

[1] Group08: Unity parkour game design - (final_1226.mp4). https://drive.google.com/drive/folders/1BGt24asCatH5SZ7rxy1M07xSZH_mtTo6. 5

[2] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 2

[3] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the IEEE international conference on computer vision*, pages 2938–2946, 2015. 2

[4] Dushyant Mehta, Srinath Sridhar, Oleksandr Sotnychenko, Helge Rhodin, Mohammad Shafiei, Hans-Peter Seidel, Weipeng Xu, Dan Casas, and Christian Theobalt. Vnect: Real-time 3d human pose estimation with a single rgb camera. *Acm transactions on graphics (tog)*, 36(4):1–14, 2017. 1, 2

[5] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. 2

[6] Yuanyuan Wang, Chao Wang, Hong Zhang, Yingbo Dong, and Sisi Wei. Automatic ship detection based on retinanet using multi-resolution gaofen-3 imagery. *Remote Sensing*, 11(5):531, 2019. 2

[7] Shih-En Wei, Varun Ramakrishna, Takeo Kanade, and Yaser Sheikh. Convolutional pose machines. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 4724–4732, 2016. 2