BEAR: Basketball Experience using Augmented Reality

Yu Chun Chen
R09922A23

Chien Cheng Chen
R09944015

Tun Chieh Lou
R09922A07

Abstract

We present an application of computer vision to generate the experience of playing basketball without a real basketball and rim. With only a laptop and a webcam, the application captures a 2.5D pose representation of the user's hand and approximates its 3D pose to render the basketball on his/her palm. Furthermore, the system detects the action of the user shooting the ball and is able to calculate the ball trajectory between the user's hand and the rim.

1. Introduction

There are a lot of ball shooting mobile apps (see figure 1.) out there, which users use their fingers to drag the ball and shoot it to the virtual rim. However, this is not the experience of people playing real basketball. People play basketball by holding the ball in their hands and shooting the ball with a specific form. With the outbreak of COVID-19, we became determined to build this application by which people are able to recall their basketball shooting experiences without going to outdoor basketball courts.

Figure 1

Before anything, the camera is calibrated, and the intrinsic parameters were stored for later use. The application pipeline works as follows: first, the camera images were captured and processed by MediaPipe, a ML library open-sourced by Google for streaming videos. Its hand solution is able to process images and return 2.5D hand pse. To render the basketball accurately, we approximate the 3D pose of the hand by its 2.5D representation. After estimating the 3D position of the keypoints, we try to search for the most likely centroid of the ball using the trilateration technique. Then, we use OpenCV to render the ball on the user's hand and the previous work we've done makes it look like the user is grabbing the ball. Also, we render the rim and make it look like it is just in front of the camera. Finally, when the user shoots the ball with a shooting form, the ball goes through a parabolic trajectory and lands on the predefined position we've set.

2. Methods

2.1 Camera Calibration

We use the camera to take about 15 photos of the traditional chess board photo. After that, we use functions in the OpenCV library to do camera calibration and get the intrinsic matrix of the camera. The camera intrinsic matrix can be applied to 3D transformation in the following sections.

2.2 Hand Pose Estimation

In this step, we want to estimate the 3D hand pose of the user so that we can do further applications with this information, like rendering the basketball, detecting shooting gestures. We use MediaPipe Hands as a high-fidelity hand and finger tracking solution. It employs ML to infer 21 2.5D landmarks of a hand from just a single frame. Besides, the pose estimation can be processed in real-time on cpu, which is a brilliant fit to our application. The MediaPipe Hands model consists of a palm detection model and a landmark detection model.

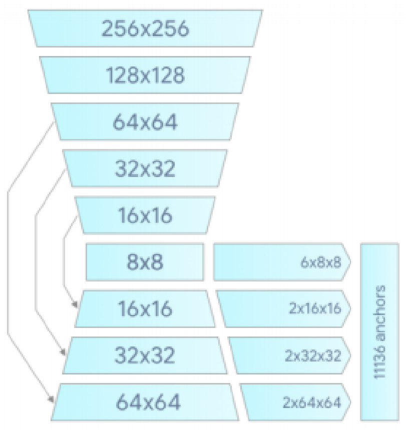

Figure 2

The palm detection model is a single-shot detector (SSD) model, which operates on a single input image and returns an oriented bounding box. The SSD approach is based on a feed-forward convolutional network that produces a fixed-size collection of bounding boxes and scores for the presence of object class instances in those boxes, followed by a non-maximum suppression step to produce the final detections. The model architecture consists of encoder-decoder feature extractor as shown in Figure 2 ,which is similar to feature pyramid network (FPN) for a larger scene-context awareness even for small objects.

After running palm detection over the whole image, the subsequent hand landmark model performs precise landmark localization of 21 2.5D coordinates inside the detected hand regions via regression. The model learns a consistent internal hand pose representation and is robust even to partially visible hands and self-occlusions. The model will output 21 hand landmarks consisting of x, y, and relative depth. The final result is shown as Figure 3.

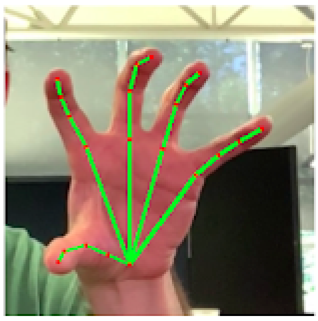
Figure 3

Finally, we refer to the paper [2] proposed method to transform 2.5D coordinates to 3D coordinates. Given a 2.5D pose, we need to find the depth of the root to reconstruct the 3D pose. Given the 2.5 pose and camera intrinsic matrix, there exists a unique 3D pose that satisfies

$$(X_n - X_m)^2 + (Y_n - Y_m)^2 + (Z_n - Z_m)^2 = C^2 \quad (1)$$

where $X_n X_m Y_n Y_m Z_n Z_m$ is the pair of normalized 3D camera coordinates of keypoints and C is the constant distance. In this project we use the C = 1 and use the landmarks of WRIST and INDEX_FINGER_MCP as the pair of keypoints, because they are the most stable keypoints in the mediapipe. The equation 1 can be rewritten as

$$(x_n Z_n - x_m Z_m)^2 + (y_n Z_n - y_m Z_m)^2 + (Z_n - Z_m)^2 = C^2 \quad (2)$$

where $x_n x_m y_n y_m$ is the pair of 2d image coordinates. Then replacing the $Z_n Z_m$ with $(Z_{root} + Z^r_n,\ Z_{root} + Z^r_m)$, and Z_{root} will become the only unknown in the equation. At last, we can use the formula solution to get the Z_{root} and calculate the scale factor based on real world length and distance in 3D camera coordinates. According to scale factor, camera intrinsic matrix and Z_{root}, we can easily reconstruct the 3D camera coordinates.

2.3 Basketball & Hoop Rendering

After getting 3D camera coordinates of the 21 landmarks of the hand, we can estimate the centroid of the ball by trilateration. In our project, we use the coordinates of thump_tip, index_finger_tip and pinky_tip to do trilateration (see equation 3), then using middle_finger_tip and wrist to verify the solution.

$$arg\ min_c \sum_n \left| ((x_n - x_c)^2 + (y_n - y_c)^2 + (z_n - z_c)^2) - r^2 \right| \quad (3)$$

In 3D objects, each vertice has 3D coordinates relative to the original point, and we can easily obtain 3D camera coordinates of the vertices by translating the original point to the centroid of the ball. Finally, we multiply these 3D coordinates by camera intrinsic matrix to get image coordinates and use OpenCV library to render the basketball. Hoop rendering is the same as basketball rendering, but we predefined the 3D coordinates of the hoop at the top of the screen. The rendering basketball and hoop are shown in Figure 4.

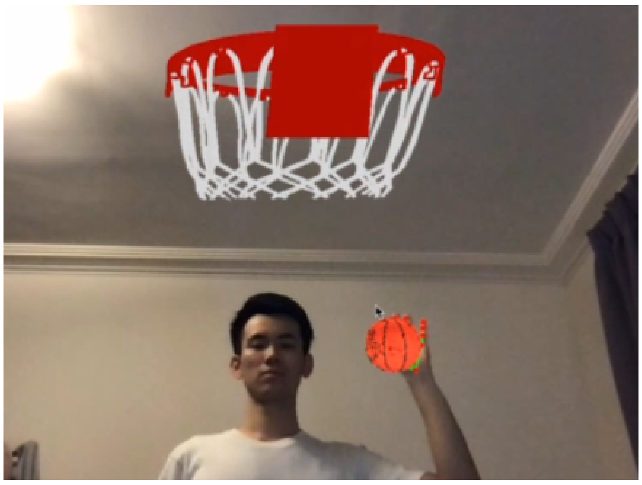

Figure 4

2.4 Shooting Detection

We use the simplest way to detect the shooting. When we finish the shooting, our hand landmark of MIDDLE_FINGER_TIP will be lower than the hand landmark of MIDDLE_FINGER_MCP (see Figure 5). Therefore, while the Y coordinate of MIDDLE_FINGER_TIP is larger than the MIDDLE_FINGER_MCP (Y becomes larger when going downwards), we consider the user shooting. This method seems simple but it turns out very effective.

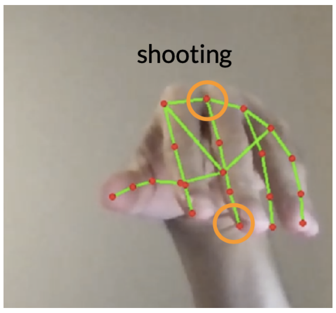

Figure 5

2.5 Basketball Movement Estimation

In order to simulate the basketball movement, we use the trajectory equation (see equation 4) to get the coordinate of the ball in time.

$$y = tan(\theta) * x - \frac{g * x^2}{2 * V_0^2 * cos(\theta)^2} \quad (4)$$

First, we take the coordinate of the ball and hoop as the start and end points, predefine the angle of shooting as 60 degrees and go with the trajectory equation to get the initial velocity. Second, dividing the velocity to horizontal and vertical and determining the velocity of x-direction and z-direction based on the horizontal distance of x and z. Finally, we can estimate the basketball movement in time by using equation (5).

$$x = v_{0x} * cos(\theta) * t + x_0$$

$$z = v_{0z} * cos(\theta) * t + z_0$$

$$y = -(v_{0y} * sin(\theta) * t - \frac{1}{2} * g * t^2) + y_0 \quad (5)$$

3. Execution

As the demo video shows, the ball is able to scale when the user's hand moves back and forth from the camera, which shows our method to render the ball is quite successful. However, the frame rate is about 4-6 per second when there is a hand in the image. This is because rendering the basketball takes time, and we used only a laptop-level CPU to run this app. When the user shoots the ball in a shooting form, our system can mostly detect it and start to calculate the trajectory of the ball.

4. Conclusion

In this work, we present an AR application that allows the users to interact with a basketball in front of a laptop. It was built of a pipeline that uses computer vision techniques as well as physics. Since it demonstrated a combination of vision and physics, I believe the result can be expanded to many other AR applications or games.

The limitations of our work is the stability. Since our application depends only on a monocular camera and a consumer laptop, it's quite hard to predict a stable and accurate hand pose with absolute depth in real-time. The unstable prediction of the hand pose will affect the rendering result of the basketball. For instance, the basketball will have unexpected jitter during the application. Despite the fact that we have added some constraints to restrict the movement of the pose between frames, the results are still unstable sometimes.

5. Work Contribution

hand detection:	Yu Chun Chen, Chien Cheng Chen
ball movement:	Yu Chun Chen, Tun Chieh Lou
rendering:	Yu Chun Chen, Chien Cheng Chen, Tun Chieh Lou
paper survey:	Chien Cheng Chen, Tun Chieh Lou

References

[1] Augmented Reality DIY.
[2] Hand Pose Estimation via Latent 2.5D Heatmap Regression
[3] MediaPipe Hand Solution
[4] 拋物線運動 - 物理化、重力加速度、力學
[5] MediaPipe Hands: On-device Real-time Hand Tracking
[6] SSD: Single Shot MultiBox Detector