BallerStats: Estimating Distance Traveled with Video Footage

Robert Buckley robuckle@umich.edu
Richard Guan guanr@umich.edu
Jensen Hwa hwaj@umich.edu
Tony Pan tonypan@umich.edu
Calvin Zheng calzheng@umich.edu

1. Introduction

Obtaining advanced statistics beyond those simply observable to the naked eye can prove immensely useful in basketball, helping teams and players glean key insights into their performance and how their movement on the court may translate to points on the board. This type of data is currently available to professional teams in the National Basketball Association (NBA). Still, it remains largely out of reach for everyday pickup basketball players who want to improve their game.

There is a company, Second Spectrum, that is officially partnered with the National Basketball Association (NBA) and provides teams with these advanced statistics to help them improve their game. This type of digital analysis would help save hours of manual review on film and provide more accurate data to coaches and players. This company currently utilizes a player tracking system and various undisclosed machine learning and computer vision techniques [1]. Similarly, a group of students from Stanford University recently completed a project to track basketball players from NCAA basketball games and map the player's positions from the court onto an aerial view of the court, showing how a player's movement changed between frames [7].

Using computer vision to obtain various advanced statistics in basketball is solvable using computer vision technologies. Our project estimates the distance a player has moved on a court, which is not an easy statistic to track while a game is occurring. This problem is solvable as it relies on object tracking, single view depth perception, and camera calibration. Object tracking has been a staple of computer vision for quite some time and is reasonably reliable. Similarly, camera calibration using checkerboards is accurate and dependable, as found in [8]. Simultaneously, the accuracy of single view depth perception has been drastically improved by deep learning techniques in recent years [6].

To address this issue, we have figured out a way to obtain this data through computer vision by tracking and identifying a single player throughout a video and then determining the real-world coordinates of the player. We can accomplish it by finding and using the intrinsic matrix of a camera and the depth of an object in an image. We obtained our intrinsic matrix through camera calibration and depth by utilizing single view depth perception. This approach then allowed us to compute and track the distance traveled by the player.

2. Approach

We developed our project in Python 3 code. To accomplish our goal of tracking the player distance, we collected video footage of a player playing basketball using a calibrated camera, drew a bounding box to identify the player in the initial frame, used a CSRT tracker to track the player through subsequent frames, applied the MiDaS model [6] to obtain a depth map of disparity to be used to estimate the depth of the player, found the real-world coordinates of the player on the court in each video frame using the depth and pixel location of the player, and computed the distance traveled over video. A visual representation of our algorithm is shown in Figure 1.

2.1. Camera Calibration

To obtain the intrinsic matrix of our camera, we calibrated a Google Pixel 3's camera following the steps outlined in OpenCV's checkboard calibration tutorial [2]. We obtained over 20 images of the same checkboard with different angles, perspectives, distances, and distortions through this process. Then, we used the intrinsic matrix obtained from the camera to map from pixel coordinates to world coordinates. The intrinsic matrix from our camera was:

$$\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \approx \begin{bmatrix} 1546.07 & 0 & 962.98 \\ 0 & 1544.42 & 526.78 \\ 0 & 0 & 1 \end{bmatrix}$$

f_x and f_y are the focal lengths, whereas c_x and c_y are the coordinates of the principal point.

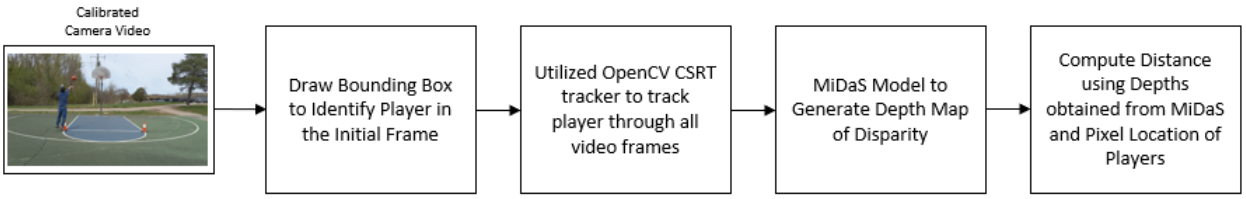

Figure 1. Algorithm overview

2.2. Player Tracking

Tracking begins once the user of our program draws a bounding box around the player and the basketball hoop in the video. We experimented with two OpenCV trackers, the MIL and CRST trackers [3, 4], to track the player in the video. The MIL tracker is based on Multiple Instance Learning as detailed in [5], but in essence, the tracker uses the space around the specified object and labels it as the object. Meanwhile, the CRST tracker utilizes the Discriminative Correlation Filter with Channel and Spatial Reliability (CSR-DCF). A spatial reliability map makes sure that the filters are constrained to only parts of the image that are supposed to be tracked [10]. This method has been known to work well with irregularly shaped objects that are non-rectangular. We ultimately chose to use CRST as it was more accurate when tracking the player in our videos, but we did note a few observations about the two different trackers. We noticed that MIL runs faster but will get lost if someone moves too quickly, whereas CRST is much slower than MIL but more reliably tracks the player. Both of these trackers work well for our single-player videos, but we must note that neither would work well for a real basketball game. The trackers would get confused when two people appear close to each other. Solving this problem involves implementing multiple object tracking (MOT), but we chose not to focus on MOT for our project.

2.3. Depth Perception

Another integral part of our project was using depth perception to help us determine the real-world coordinates of the player. For depth perception, we used the MiDaS model to generate depth maps of disparity for each of the frames in our video [9]. The MiDaS model was beneficial for our object as it was developed for monocular depth estimation, allowing us to obtain the relative depth of the player from the camera in each frame. MiDaS is relatively accurate when used for single-view depth perception as it combines deep learning techniques and a large variety of datasets, including 3D movies [9]. This model was then evaluated on datasets not used in training, in which the results were very accurate [9]. For our project, we slightly modified the input of the MiDaS model so that MiDaS takes in videos as inputs and outputs depth disparity maps as images. With these disparity maps, we obtained the disparity matrix of the image, which we then used to compute the depth and, ultimately, the distance.

2.4. Distance Computation

To compute the distance traveled by the player, we referenced all of the data we collected through the drawing of the bounding boxes, player tracking, camera calibration, and depth perception. Recall that two bounding boxes are drawn in each video: one over the player and the other over the basketball hoop. We decided to use the basketball hoop as our landmark as it was simple to obtain the distance to the camera. This technique worked well, as we could use this to find the scaling constant to convert our pixel coordinates into real-world ones. From the depth perception part of our project, we obtained both the disparity matrices from the basketball hoop and the player. We calculated the depth by using the following equation: $D = dH * dispH/dispP$, where D is depth, dH is the known depth from the hoop to the camera, $dispH$ is the disparity of the hoop, and $dispP$ is the disparity of the player. The disparity of the player and the hoop was obtained by using the pixel values of the player and hoop from the object tracker as the indices in the disparity matrix of the image. With the depth of the player, we computed the real-world of the player by using the following equations below, where x and y are the real-world coordinates of the player, x_1 and y_1 are the pixel coordinates of the player, (c_x, c_y) is the principal point obtained from the intrinsic matrix, and f_x and f_y are the focal lengths from the intrinsic matrix.

$$x = (x_1 - c_x) * D/f_x$$
$$y = (y_1 - c_y) * D/f_y$$

From this, we obtained the real-world coordinates of the player in each frame, which we then used to compute the Euclidean distance of the player and the total distance traveled over the duration of the video.

3. Experiments

To ensure the validity and accuracy of our project, we tested various parts of our project in isolation and then as

a whole. We evaluated our camera calibration system and depth estimation separately before testing the overall accuracy of the distance computed.

3.1. Data Collection

To complete this project, we needed to record our videos to simulate what it would be like for pickup basketball players to videotape themselves practicing or playing a game. We went to a nearby park in Ann Arbor, Michigan, and took several measurements, including the distance from the basketball hoop to our camera and from the hoop to the free-throw line. Figure 2 showcases all of the measurements we took of the basketball court. For the videos, we first recorded the player running in straight lines along these measurements before advancing to more freeform paths.

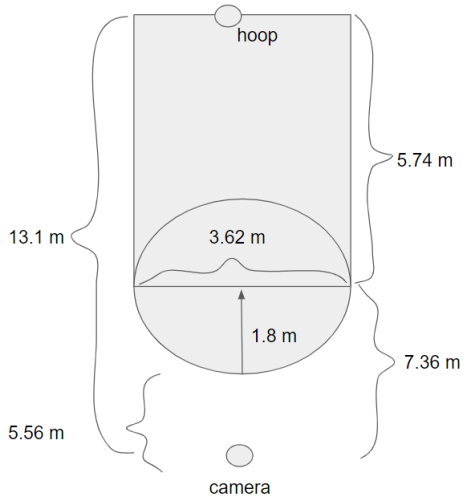

Figure 2. Leslie Park Basketball Court Measurements

3.2. Empirical Evaluation of Camera Calibration

As we finished writing the code for camera calibration, we wanted to ensure that it was valid; therefore, we tested it by performing an extrinsic calibration with April Tags [11]. Below, Figure 3 showcases our setup for evaluating our camera calibration.

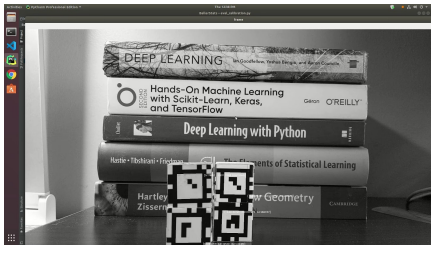

Figure 3. Setup for Extrinsic Calibration with April Tags

After performing the extrinsic calibration, we compared the values we obtained with the actual measurements. Table 1 below shows the results of this evaluation.

Table 1. Results of April Tags Evaluation

Tag No.	Ground Truth (m)	Predicted Value (m)	Absolute Diff. (m)	% Error
0	[0.2, 0.0127, 0.0381]	[0.1784, 0.0103, 0.0474]	0.0237	11.59%
1	[0.2254, -0.0127, 0.0381]	[0.2033, -0.0119, 0.0478]	0.0242	10.55%
2	[0.2, 0.0127, 0.0127]	[0.1769, 0.0107, 0.0253]	0.0264	13.15%
3	[0.2254, -0.0127, 0.0127]	[0.2016, -0.0115, 0.0257]	0.0271	12.00%

From this evaluation, we observed that the average percent error among the four tags was 11.78%. We decided to use absolute average percent error as our error evaluation metric due to there being multiple measurements. We expected the errors to have a mean of zero, and the absolute error difference here is in the range of 0.2 meters that would not reflect well onto other ranges. Although our error is 11.78%, the actual error is much lower since most of the error was caused by external factors. These external factors included the camera not being perfectly perpendicular to the surface or the April Tags not being precisely at the specified locations.

3.3. Evaluation of Depth Estimation

After applying the MiDaS model to our video, we took random image frames, computed the depth, and compared it with our ground truth values. Examples of depth map images returned by the model are shown in Figure 4. Our evaluation results are shown in Table 2.

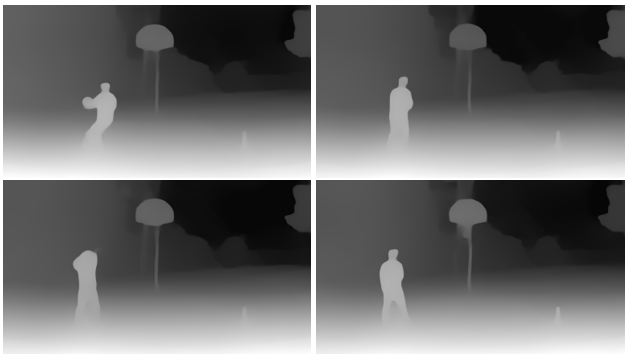

Figure 4. Depth Map Images

From our results, we gathered that the average absolute error among all of the images was 1.17236 meters while the

Table 2. Depth to Player Evaluation

Image Name	Ground Truth (m)	Predicted Value (m)	Absolute Diff. (m)
short1/0	13.10	11.62808	1.47191
short1/7	7.36	7.61437	0.25437
short2/0	13.10	11.77528	1.32471
short2/6	7.36	8.19754	0.83754
short3/0	13.10	7.43173	5.66826
short3/6	5.56	4.75771	0.80228
med3/0	13.10	13.22718	0.12718
med3/6	7.36	6.51299	0.84701
med4/0	13.10	12.75974	0.34025
med4/6	7.36	7.30994	0.05005

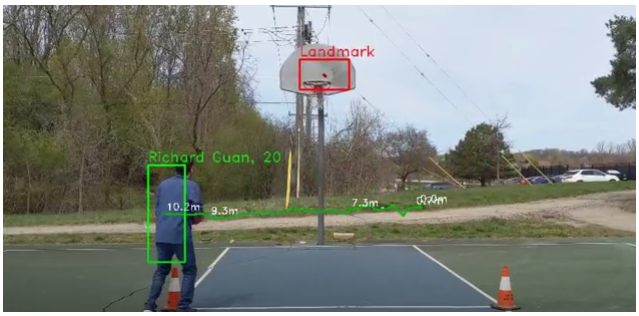

Figure 5. Visualization of Distance Traveled

average percent error was 11.67%. These results were not wholly unexpected as depth perception based on a single camera is difficult; therefore, we expected many noise and inaccuracies from depth estimation.

3.4. Distance Evaluation

The distance evaluation is essentially the final evaluation of our project, as our main goal was to track player distance using a video. To help us evaluate and better showcase the distance, we added visuals on each image frame to showcase where the player has been and the distance traveled, as showcased in Figure 5. Once again, we compared the distances computed for each video by our algorithm to the ones that we measured, and the results are shown in Table 3.

We computed the average error to be 4.936 meters and the average percent error to be 46.146%. Given that there was some error in both the camera calibration and depth perception stages, it is not surprising that there is a decent amount of error in our final estimation. We should note that our distances are still relatively accurate.

4. Implementation

Our project comprises many different parts: camera calibration, player tracker, depth perception, translation to real-world coordinates, and data visualization. For our camera

Table 3. Distance Traveled by Player Evaluation

Video Name	Ground Truth (m)	Predicted Value (m)	Absolute Diff. (m)	Percent Error
short1	5.74	7.95	2.21	38.50%
short2	5.74	9.50	3.76	65.54%
short3	7.54	8.39	0.85	11.27%
med1	7.24	10.88	3.64	50.28%
med2	7.24	10.36	3.12	43.09%
med3	7.46	11.96	4.50	60.32%
med4	7.46	11.85	4.39	58.85%
long1	41.18	58.20	17.02	41.32%

calibration section, we followed the procedures based on OpenCV's Checkboard Calibration tutorial [2]. When evaluating our calibration, we calibrated the extrinsics of the camera using April Tags, which returns the pixel coordinates of the tags [11]. We computed the world coordinates of each April Tag using both intrinsics and extrinsics camera parameters. We compared that with the ground truth tag locations. For the player tracker, we utilized the CSRT tracker from OpenCV [3] based on [10] but wrote the code to draw the bounding boxes and give visual indications of what was selected. In terms of depth perception, most of the code in the depth folder was imported through the MiDaS repo [9] to use the model on our code. We modified the original script that runs the model to take either photos or videos as input. We wrote all of the code relating to the translation of image pixel values to real-world coordinates. We created a visualization on the final frame to showcase where the distance was computed by showing the player's trajectory.

References

[1] "Second Spectrum - Our Work." Second Spectrum. https://www.secondspectrum.com/ourwork/teams-leagues.html (accessed Apr. 24, 2021). 1

[2] "Camera Calibration." OpenCV. https://docs.opencv.org/master/dc/dbb/tutorial_py_calibration.html (accessed Apr. 24, 2021). 1, 4

[3] "cv::TrackerCSRT Class Reference." OpenCV. https://docs.opencv.org/3.4/d2/da2/classcv_1_1TrackerCSRT.html (accessed Apr. 20, 2021). 2, 4

[4] "cv::TrackerMIL Class Reference." OpenCV. https://docs.opencv.org/3.4/d0/d26/classcv_1_1TrackerMIL.html#details (accessed Apr. 20, 2021). 2

[5] B. Babenko, M.-H. Yang, and S. Belongie. Visual tracking with online multiple instance learning. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 983–990, 2009. 2

[6] B. Babenko, M.-H. Yang, and S. Belongie. Robust object tracking with online multiple instance learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(8):1619–1632, 2011. 1

[7] E. Cheshire, M.-C. Hu, and M.-H. Chang. Player tracking and analysis of basketball plays. 2015. 1

[8] A. Datta, J.-S. Kim, and T. Kanade. Accurate camera calibration using iterative refinement of control points. In *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pages 1201–1208, 2009. 1

[9] K. Lasinger, R. Ranftl, K. Schindler, and V. Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *CoRR*, abs/1907.01341, 2019. 2, 4

[10] A. Lukezic, T. Vojir, L. Cehovin Zajc, J. Matas, and M. Kristan. Discriminative correlation filter with channel and spatial reliability. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 2, 4

[11] J. Wang and E. Olson. AprilTag 2: Efficient and robust fiducial detection. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2016. 3, 4