CS231A Course Project - Monocular 3D Baseball Swing Tracker

Anonymous submission

Paper ID

Abstract

This paper addresses the research behind monocular 3D reconstruction and it's application to baseball. Given a video of baseball player swinging a baseball bat, we attempt to reconstruct the 3D coordinates of the bat path as the bat moves through space. After training a YOLOv5 model on a large sample of baseball bats in various angles and states, we attempt to detect the bounding box of the baseball bat, and further dissect the area within the bounding box to further carve out the shape of the bat using methodologies such as Gaussian Blurring, Canny Edge Detection and Hough Line Transforming. A fully connected neural network that is trained on biomechanical swing data obtained through OpenBiomechanics released by Driveline Baseball [9] is then introduced to estimate the 3D position of the 2D trajectory of the swing path.

1. Introduction

In almost every sport these days, we are able to capture every movement that happens on the field, court or ice, through optical tracking and vision processing systems that are implemented throughout venues across the world. Implementing these systems is an expensive endeavor, in which only organizations with viable means such as professional sports organizations or large universities are able to implement. In this paper, I outline my attempt to bring the reconstruction of captured sport movement, in this case a baseball swing, to the everyday user through monocular 3D reconstruction. A lot of research has been done over recent years to reconstruct golf swing mechanics [3] [7] [1]. [8] Found pairs of parallel line segments to detect golf club and utilized dataset from CMU to make educated guesses on how the golfer will move. They then utilized a least squared approach to refine guesses throughout the motion of the swing. Utilizing the 3D motion data of swings is something that I also explore, except using baseball swing data and instead using a deep learning approach. Similarly [1] utilized canny edge and hough line transformation to detect lines, and use position of hands to detect the golf club head. This is instructive in my research, as I detect hand locations to approximate both the bat knob and bat head locations. Once swing paths are generated, 3D lifting is needed to create 3D swing trajectories. [6] explored 3D position estimation based on 2D images as it relates to soccer ball trajectories. The research explores a deep learning approach in terms of calculating relative sizes and position of soccer ball to estimate 3D position. Leveraging relative positions of the bat throughout the trajectory of swing is something I explored in the model I introduce as well. Even though it is not an approach I ended up going with, utilizing their approach for using relative positions in a model to estimate depth is something that I ended up implementing as well. [5] also described a 2D to 3D lifting approach as it pertains to human pose estimation. While it is ultimately a different problem they are trying to solve by attempting to create more robust deep learning 3D lifting techniques, it helped shape my understanding of the different routes I can take to model a 3D baseball swing by only using 2D information.

2. Problem Statement

The first piece of being able to reconstruct a baseball swing from a single video is to understand where the bat is located in each frame of the video. There has been a lot of research done in the sport of golf to reconstruct golf paths [3] [7] [1]. I will apply similar methodologies, as well as introduce YOLO detection in order to further improve the ability to detect where the bat lies on the image, while distilling the reconstruction of the 3D bat path to just the detected area. In order to predict the location of the bat throughout the video, I would need a large sample of baseball bat images to train a model on. Detecting the baseball bat could come in a few different forms. Similar to pose estimation, where joints are points on the body that are estimated and connected to one another, it can be argued that a bat is also made of two points:

1. **Bat Knob**
2. **Bat Head**

Connecting these two points theoretically make up that bat, or atleast the part of the bat we are interested in, which is just the straight line coordinates. However, detecting a bat from a single video is more challenging when trying to estimate bat heads and bat knobs. As can be seen in figure 1, rarely do bat knobs or bat heads show up cleanly in images, and thus make it difficult to detect. In monocular bat tracking, it seems rather unlikely that one can estimate a bat from a video by attempting to detect both the bat head and bat knob individually.

Figure 1. Displayed difficulty of determining bat head/bat knob in Front View

Another approach is polygon segmentation. A baseball bat has a rather unique shape, and being able to capture the body of the bat without worrying about filtering out the background is ideal for accurate detections. However, carving out polygons is a time consuming process, and while it could potentially lead to more accurate results in the future, it seems unlikely that the amount of images that would be required to carve out polygons would be beneficial in the scope of this project given the number of images that would be needed to manually segmented. There are a few different publicly available datasets online with labeled bat images for object detection purposes. Ideally the dataset chosen for this project has bat labeled images from a variety of different angles, captured via different mediums. I've landed on utilizing data from [2] which has over 8000 labeled bat images, captured either via broadcast television, social media, or mobile phone.

Detecting the bounding box where the bat is located is only the first step of the process. Figure 2 illustrates the general pipeline I utilized for this process. Each piece will be touched on in more depth in the technical approach section.

3. Technical Approach

The technical approach I'm proposing is hierarchical by nature, in terms of first detecting a region of interest within an image frame, and from there further dissecting the bounding box to retrieve the coordinates of interest. In the scope of this project, I will mainly be focusing on the front view angle which is represented in figure 2, given its the angle that's the most useful for a hitter to analyze with regards to swing path. In this section I will go deeper into the methodology behind each step of the pipeline.

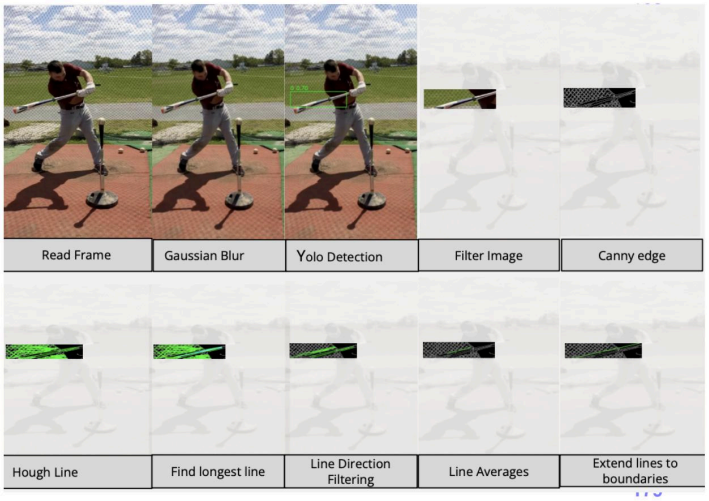


Figure 2. Bat Detection Pipeline

3.1. YOLO Object Detection Model

The first step as mentioned above, is detecting areas on each frame of the video where a bat is located. In the midst of a swing, a baseball bat is represented in various angles. To detect a bat, I used a labeled dataset [2] which has labeled boxes on roughly 8000 images spanned across many different forms of media. Given a bat tracking tool would be most heavily utilized via a user capturing video via their phone, this dataset was chosen due to the unique variety of images pulled anywhere from MLB broadcast video, to video screen captured from social media. The variety of angles provided also helps capture a lot of the noise that a single swing can provide. To train our YOLO model for object detection, I followed a structured approach. Within the scope of this project, I didn't pursue further performance tuning on this Yolo detection model, given the importance of being able to develop methodologies to detect the bat without the reliance of perfect detection rates given by the model itself. However, I would like to further enhance the dataset used for this model in future research, as well as explore ways to tune the model for better future performance. The process to train the model went as follows:

1. **Data Gathering**: Determine sufficiency of public data for this project. Roboflow has a number of baseball related labeled datasets. I found one that seemed robust enough for what I required.

2. **Data Transferring**: I accessed the dataset stored on Roboflow via a GPU enabled VM on Google Cloud Platform

3. **Training**: After installing pytorch, ultralytics and

other necessary modules, I began training the custom detection model. Using pre-trained weights as a starting point, the model was trained with a batch size of 16 for 10 epochs.

4. **Evaluation**: Model performance was evaluated on a validation set by looking at the F1 curve, a metric evaluating it's precision and recall.

As shown in 3 the F1 curve for the YOLO bat detection model, class 0, which represents the bat, is pretty robust from confidence intervals .15 to .75, peaking at a confidence interval of about .5. The hand model, class 2, performs just below that level, however in this process it is not as important as the bat detection model in terms of creating quality 3D animations since we can make some assumptions about hands based on other information. In both cases however, adding images and performing fine tuning on the model seems like it could make a big difference in terms of performance in future research.

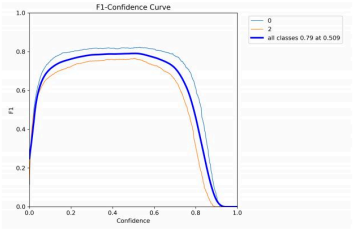
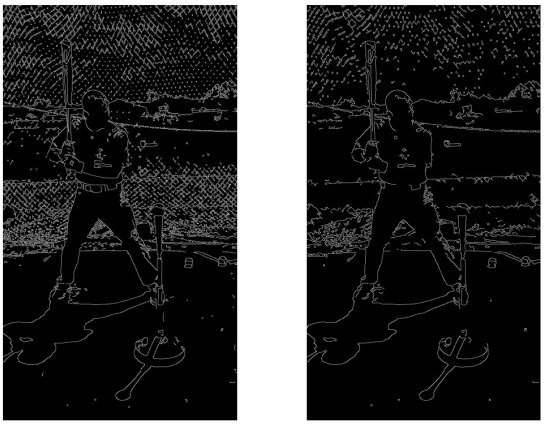


Figure 3. Class 0 (Bat) curve showing peak around .5

3.2. Noise Reduction Techniques

It is rare to take video of someone hitting a baseball without there being noise in the background. Noise can come in many different forms, but when it relates to hitting, there are many instances where noise can come from netting from batting cages, background crowds, or players. Because of the importance of removing noise to zero in on the object of interest, I explored different ways we can remove some of that background noise before running the image through the Yolo detection model and line detection process. Bilateral filtering and Gaussian filtering are two of the main image smoothing techniques I attempted in my pipeline. Through testing, I noticed that Gaussian Filtering was much more performant in terms of object detection and line detection of interest compared to Bilateral filtering. As explained in [4] Gaussian filtering takes a weighted-average approach to smoothing pixels at a local level. Bilateral filtering on the other hand takes into account pixel intensity, which results in edges being preserved more robustly in the image compared to the Gaussian filter. While I would have hypothesized that the more profitable approach in this case would be to apply Bilateral filtering so the shape of the bat is more intact before running the canny edge detector, I noticed that in most cases, Gaussian filtering keeps the structure of the bat intact enough, while removing much of the unnecessary edges in the background. Figure 4 shows an example of Bilateral filtering retaining much of the unnecessary edges in the background, while Gaussian filtering maintains the shape of the bat, while removing much of those edges. The Gaussian filter example should show the importance of noise reduction, as it is likely that the line detection process would lead to a lot of inaccurate bat predictions given the number of lines represented in the image without any sort of smoothing.



(a) Bilateral filtering example (b) Gaussian filtering example

Figure 4. Bilateral Filtering vs. Gaussian Filtering

3.3. Canny Edge Detection and Hough Line Transform

The next step in the pipeline is to apply canny edge detection and hough line transformation. Canny edge detection is applied within the bounding box, resulting in a number of edges within the cropped box within the image. I hypothesize that within the bounding box in which the bat is detected in, the longest detected line is where the bat is located. I do this by applying hough line transformation to find the longest line. As shown in figure 5, the importance of pre-filtering the image down to the bounding box before detecting the straight lines on the image. Otherwise, unless the image has very little noise in the background, it would be difficult to predict bat location based off of lines as seen in the non filtered image, where the longest line is shown to be the shadow on the ground. Once the longest line is detected, neighboring lines are found based on the direction of the longest line. The neighboring lines that fit within a custom threshold of the longest line's direction remain as data points to hedge the chance that the longest line is not

the bat. The mean of the locations of the filtered longest lines is taken, and gives the approximate bat location in the frame. We then expand the line to the points in which it intersects with the bounding box to give the full bat length in the frame.

(a) Detected box area

(b) Not filtering to detected box

Figure 5. Showing importance of object detection to filter image to detected bounding box before edge detections.

Imputing Missing Bounding Box Locations

It is important not to completely rely on the Yolo Bat Detection model when it comes to filtering the frame down to the region of interest in the image. Because of the importance of subsetting the image, so as we have a greater likelihood of a positive bat line detection, we implement a simple methodology to impute the next bounding box based on the previous bounding boxes' directions and locations.

In the following example, w represents the width of the bounding box, h represent the height of the bounding box, while x and y represent the respective starting points of the bounding box. We first begin by finding the coordinates of the previous two bounding box locations and calculating the difference in the x and y locations of the two bounding boxes:

$$\begin{aligned} &(x_{n-1}, y_{n-1}, w_{n-1}, h_{n-1}) = \text{Frame}_{n-1} \\ &(x_{n-2}, y_{n-2}, w_{n-2}, h_{n-2}) = \text{Frame}_{n-2} \\ &\Delta x = x_{n-2} - x_{n-1} \\ &\Delta y = y_{n-2} - y_{n-1} \end{aligned} \tag{1}$$

We then apply the difference in x and y locations to the width and height of the previous frames bounding box location:

$$\begin{aligned} &w_{n-1} \leftarrow w_{n-1} + |\Delta x| \\ &h_{n-1} \leftarrow h_{n-1} + |\Delta y| \\ &x_{n-1} \leftarrow \max(x_{n-1}, 0) \\ &Frame_n \leftarrow (\lfloor x_{n-1} \rfloor, \lfloor y_{n-1} \rfloor, \lfloor w_{n-1} \rfloor, \lfloor h_{n-1} \rfloor) \end{aligned} \tag{2}$$

This methodology results in a bounding box that is a bit larger than previous bounding boxes, since we are applying the direction magnitude to the previous bounding box's width and height. The more consecutive missed bounding box detections results in larger and larger imputed bounding boxes, given the additive nature of this approach. However, this methodology was chosen because of two main reasons.

1. The difference in bat angles from frame to frame is highly dependent on the frame rate of the camera. While I explored applying the direction vector of the previous two bounding boxes to the previous bounding box, thus giving a smaller area to detect lines within, there is a greater likelihood that the next bounding box misses the bat entirely, especially with lesser frame rate cameras.

2. In instances where the bat detection model misses in consecutive frames, the imputed bounding box will stall in direction, and most likely again miss the bat entirely for multiple frames if the Yolo detection model fails to detect the bat.

3.4. Bat Point Labeling

For us to take meaningful insights out of 3D bat positions, understanding which of the line endpoints is the bat knob and which endpoint is the bat head is important. During the detection process, bat knobs and bat heads are estimated based on location to center of the frame.

We use the distance formula between two points:

$$\text{distance}(x, y, c_x, c_y) = \sqrt{(x - c_x)^2 + (y - c_y)^2} \tag{3}$$

For the two endpoints of the line, we follow the following process:

Calculate the center of the image:

$$c_x = \frac{\text{width}}{2}, \quad c_y = \frac{\text{height}}{2}$$

Calculate distances from the center:

$$\text{distance}_1 = \sqrt{(x_1 - c_x)^2 + (y_1 - c_y)^2}$$

$$\text{distance}_2 = \sqrt{(x_2 - c_x)^2 + (y_2 - c_y)^2}$$

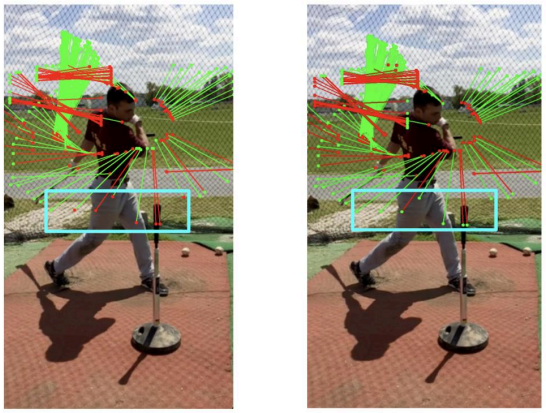
The endpoint further from the midpoint is labeled as the bat head

$$\begin{aligned} &\text{If distance}_1 > \text{distance}_2 : \\ &\quad (x_f, y_f) = (x_1, y_1) \\ &\text{Else:} \\ &\quad (x_f, y_f) = (x_2, y_2) \end{aligned} \tag{4}$$

This assumption, however, that the bat head will always be further from the midpoint of the frame is a bit presumptuous. The reason the midpoint of the frame is initially chosen

as the anchor point, is because for a video to capture a bat in totality in a video from the side angle, the bat head will more likely be operating outside of the middle of the frame. As shown in show figure there are times where, especially if the bat is operating on a flatter plane closer to the middle of the frame, the bat head will be closer to the midpoint.

After the detection process is over, we store all of the line detection coordinates of the bat, and hand detection coordinates. Because the knob of the bat should always be closer to the batter's hands, we run the same distance equation as above for each endpoint, but use the hand detection coordinates as the anchor point. Because the hand detection model isn't quite as robust as the bat detection model, we leverage the hand coordinates from the frame closest to the current frame to calculate the distance from.



(a) Bat head detections before running through hand distance method (green dot equals good head detection, red dot equals bad detection)

(b) Bat head detections after running through hand distance method

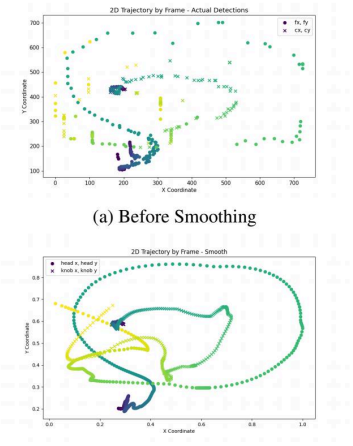
Figure 6. Bat detections (red dots indicate false bat head detections, while red lines indicate lines detected by imputed bounding box methodology

3.5. Interpolation

While the YOLO bat detection model is a good starting point, it is unreasonable to assume that it will be able to detect the bat in every frame of the video. Between shadows, blur, odd angles, and any other random noise, the bat will likely go undetected in a number of frames. This could partly be improved if the model is fed more data, although it will likely never be perfect. We can assume, however, that as long as there are enough detections in the video, we can apply linear interpolation, or some form of interpolation as is explored with regards to motion analysis in [3], to predict the bats next location in the frame, given some sample before and after the current image frame.

3.6. Smoothing

While the bat detection process does a relatively good job detecting a line that represents the bat in a frame, it is inexact due to noise in the image. Due to the inexactness of both bat directions and bat lengths detected across the frames, smoothing is applied to the time series data in order to remove some of the false detection noise. In specific, Savitzky-Golay filter is implemented as it is a form of smoothing that operates within windows of time and applies a best fit polynomial curve to the data that helps it retain the correct shape, while removing noise. While figure 7 also includes interpolation in the second image, it can be seen the effectiveness of the Savitzky-Golay filter.



(a) Before Smoothing

(b) After Smoothing (And interpolation)

Figure 7. Before vs. After Savitzky-Golay Filtering

3.7. 2D to 3D mapping

OpenBiomechanics [9] is a project released by Driveline Baseball to make baseball biomechanics data open source to help drive innovation in the space of baseball biomechanics, as well as the health space overall. As part of the OpenBiomechanics project, they released markered captured swing data that has key points for each part of the body, as well as on the bat itself. Figure 8 shows an example of the markered bat in the data. The bat has three markers of interest as it relates to this project. Marker1 represents that bat knob, while Markers2 and Markers3 live on the bat head. Because the output of the bat detection pipeline introduced in this paper outputs a 2D line in each frame, we use the midpoint of Marker2 and Marker3 for the bat head, and keep Marker1 as the bat knob.

Within this dataset includes hundreds of biomechanics hitting sessions, with multiple swings per session. I hypothesized that using the markered bat motion data, I could esti-

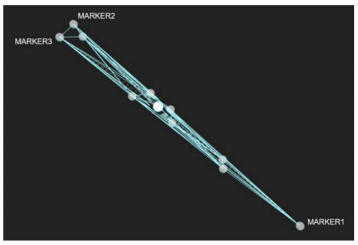


Figure 8. Markered bat provided by Driveline Baseball. Marker1 represents bat knob and midpoint between 2 and 3 represents bat head for this project.

mate depth based on the 2D coordinates. This means using the 2D trajectory of the bat path to estimate depth at each timestamp. As can be seen in Figure 9, two swings showing the similarities in 2D swing trajectories from a swing in the OpenBiomechanics dataset, and one captured via the bat detection pipeline introduced in this project. There are obvious subtle differences between the two trajectories, however, this is more likely due to imperfect detections as a result of the pipeline introduced in this paper, than it is a systematic difference in methods of capture.

Given the general similarities between the 2D trajectories, I hypothesized that if we can create a model that can predict depth on the data in the OpenBiomechanics dataset given the 2D trajectory, we can apply that model to the one captured via the detection pipeline. To do so, it is important to first normalize all coordinates so they are on the same scale.

Let D be the data frame with dimensions $m \times n$.

1. Flatten D into a single column vector F:

$$F = \text{reshape}(D, [m \times n, 1])$$

2. Normalize F using min-max scaling:

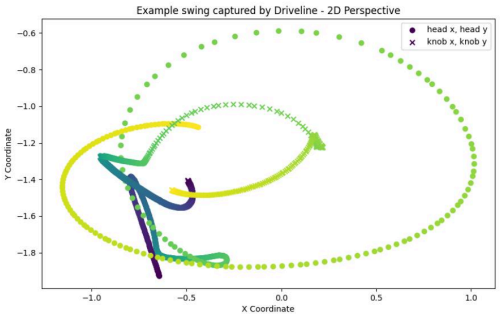
$$F_{\text{norm}} = \frac{F - \min(F)}{\max(F) - \min(F)}$$

3. Reshape F_{norm} back to the original dimensions of D:

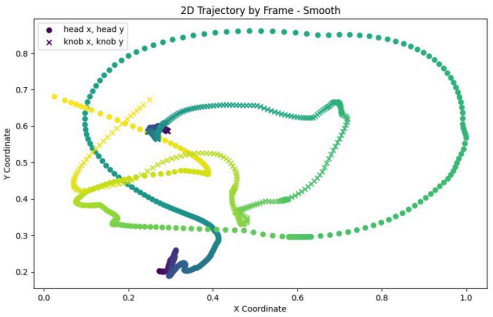
$$D_{\text{norm}} = \text{reshape}(F_{\text{norm}}, [m, n])$$

4. Return the normalized data frame D_{norm}.

Flattening the x,y,z variables into one column vector before normalizing allows the trajectories to retain the same shape and distance. Otherwise, the relationship between the x and y variables become distorted, thus altering the trajectory. After we normalize the variables, a small bit of feature engineering is implemented. In the process outlined below, Marker1 will represent the bat knob, while Marker23 will



(a) Random 2D Swing Trajectory provided by Driveline baseball's Opoen-Biomechanics Dataset



(b) 2D Swing Trajectory as a result of iPhone video that ran through bat detection pipeline

Figure 9. Similarities in swing trajectories between the two methods of capture

represent the bat head as it is the midpoint of Marker2 and Marker3. In this data, Y is the depth variable we are trying to predict for both Marker1 and Marker23. For data simplicity in this project, as we are just testing the feasibility of capturing the depth of a swing using the 2D trajectory, we filter the biomechanics data to just right-handed batters. Without having thought through the subtle differences between training a model with both right handed hitters and left-handed hitters, error would likely be introduced, and it's something that I'd love to continue thinking through in future research. Perhaps it's as simple as flipping the depth coordinate so it's the same as the right handed hitter before normalizing, but that will be explored at a later date.

To predict Marker1Y and Marker2_3Y, we will use the variables Marker1X, Marker1Z, Marker2_3X, and Marker2_3Z. We will also consider their values at the previous step $n - 1$ and the next step $n + 1$, with the idea being that knowing relative 2D spatial positions before and after the current frame will represent strong features in the model. A neural network model was built to predict the

Y coordinates of the markers. The model consists of three dense layers with ReLU activation functions, which we'd hope captures the complex nature of depth positioning as it relates to the bat's trajectory at any moment in time. The model was trained for 100 epochs with a batch size of 32. A validation split of 20% was used as well. Using Mean-Squared Error as the loss function, we achieved a value of .0086, which signals that depth of the 2D swing trajectory is captured via this model, although at this point it's a bit hard to quantify the measure at which this number is good or bad given subtle differences in swing trajectories can lead to large differences as a result. For reference figure 10 shows the predicted swing trajectory based on the features described above, compared to the actual swing trajectory. In general, it is not as smooth as the actual swing path, however, moothing techniques have yet to be applied in this example.

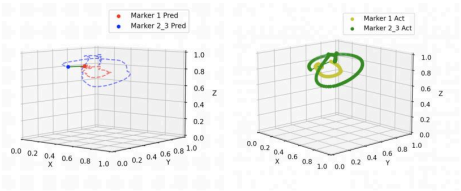


Figure 10. Predicted vs. Actual Swing from Biomechanics data based on 2D Trajectory

Finally, in figure 11, it is shown a 3D trajectory of swing path taken via an iPhone as an example for this project. At first glance, much of the noisiness seems to be due to some false line detections, however we can see that this provides a starting point to continue to work off of.

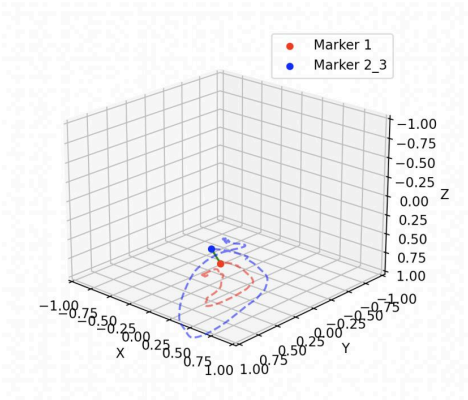


Figure 11. Predicted 3D Swing Trajectory based on Monocular Video Capture

3.8. 3D Animation Rendering

After the 2D to 3D mapping takes place, we are able to render 3D animations of the 2D trajectory. In this project, I utilized matplotlib within python for such animations as it was easy to see output as the ending result of a detection pipeline run, however in future research, creating a more robust frontend to capture video and watch the animation on a mobile device through other methods is desired.

4. Conclusion

With most results in this project being qualitative by nature, a natural next step would be to turn these results back into real world scale so we are able to draw better conclusions from it. There are also a lot of areas and methodologies within this research that should be refined and readdressed as well. The main ones that come to mind are:

1. Fine tuning and creating a more robust bat/hand labeled dataset
2. Create a more analytical approach to detecting bat line after hough line transformation step
3. Outlier detection of lines would be beneficial to implement before Savitzky-Golay filtering
4. Make modeled approach scalable enough for lefty hitters, as well as incorporate more robust biomechanics dataset
5. Do more feature engineering and test a wider set of modeling techniques for 2D to 3D mapping
6. Create a nicer looking frontend for animations

These are just some of the steps that come to mind that can be taken to make this process better in the immediate future, while longer term approaches such as bringing such a process to a user's phone would be an ultimate long-term goal.

- Link to code:github repo
- Link to full animation:Animation Link

References

[1] Ravi Chugh. Golf report: Analysis and insights into golf swing mechanics. Technical report, University of Chicago, 2023. Accessed: 2023-05-17. 1

[2] FP. Baseball batch 1 dataset. https://universe.roboflow.com/fp-kzbbo/baseball-batch-1, sep 2023. visited on 2024-05-18. 2

[3] C. K. Ingwersen, J. N. Jensen, M. R. Hannemose, and A. B. Dahl. Evaluating current state of monocular 3d pose models for golf. In *Proceedings of the Northern Lights Deep Learning Workshop*, volume 4, 2023. 1, 5

[4] Sylvain Paris, Pierre Kornprobst, JackTumblin Tumblin, and Fredo Durand. Bilateral filtering: Theory and applications. *Foundations and Trends® in Computer Graphics and Vision*, 4(1):1–75, 2008. 3

[5] V. Patel, J. Chen, and A. B. Koranteng. Lifting 2d keypoints to 3d human pose estimation. *Stanford University*, n.d. Stanford University [Journal-article]. 1

[6] Filippo Pedrazzini. Estimation of the 3d position of a ball from 2d videos using deep learning techniques. Master's thesis, KTH Royal Institute of Technology, 2019. 1

[7] E. Perini. *Feasibility of Mobile Phone-Based 2D Human Pose Estimation for Golf: An analysis of the golf swing focusing on selected joint angles*. Dissertation, KTH Royal Institute of Technology, 2023. 1

[8] Raquel Urtasun, Leonid Sigal, David J. Fleet, and Aaron Hertzmann. Learning concise models of human motion from stochastic motion capture. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8. IEEE, 2005. 1

[9] Kyle W Wasserberger, Besky David M Jones BR Brady, Anthony C, and Boddy Kyle J. The openbiomechanics project: The open source initiative for anonymized, elite-level athletic motion capture data. 1, 5