DEVELOPING AN OBJECTIVE REFEREEING SYSTEM FOR FENCING: USING POSE ESTIMATION ALGORITHMS AND EXPERT KNOWLEDGE SYSTEMS TO DETERMINE PRIORITY AND ENSURE FAIRNESS

Haokai Zhou[1], Aleksandr Smolin[2]

[1]Tarbut V' Torah Community Day School, 5200 Bonita Canyon Dr, Irvine, CA 92603
[2]Computer Science Department, California State Polytechnic University, Pomona, CA 91768

ABSTRACT

Fencers in foil and sabre are often concerned with their referees' preferences when determining priority, which determines who receives the point in a bout [1]. Oftentimes, humans fail to rationally determine priority and apply the rules fairly, leading to inconsistencies in decisions in the same bout. This often causes heated arguments and much discord in fencing competitions [2].

This paper develops software to identify fencers on a video recording, locate key points in their body's structure, record their movements and critical metrics about their performance, and match them with an objective expert knowledge system in order to determine who truly has priority at any given time in the match. We tested out several pose estimation algorithms, such as Yolov5, Yolov7, and MediaPipe in order to determine which one has better accuracy and performance in order to be able to deliver precise, unbiased, and fair refereeing decisions in a short period of time, and then allow the referees to reference the logic behind the decision, as well as see all the data that the decision was based upon in order to validate its veracity [3][4]. We also use caching technology to be able to quickly reload and review previous decisions in case any doubt about the bout's outcome arises post-fact.

KEYWORDS

Python, Yolov7, OpenCV, Fencing

1. INTRODUCTION

In foil and sabre fencing, the concept of right-of-way, or priority, exists to determine which person will get the point in the common situation where both fencers successfully land a valid hit on target [5]. The person who has priority is the one who is attacking, which often means to be moving towards their opponent while their opponent is retreating, however, if the person's march forward is too slow, it will be considered preparation and not an attack. The fine line between preparation and attack is extremely minute, with no specific definition of it in the rules of the International Fencing Federation which is the official governing body of the sport [6]. In the rule books, the only definition of an attack is that it must be committed within the "fencing time",

David C. Wyld et al. (Eds): ARIN, CSITA, ISPR, NBIOT, DMAP, MLCL, CONEDU -2023
pp. 103-114, 2023. CS & IT - CSCP 2023 DOI: 10.5121/csit.2023.130710

which has no objective definition. The interpretation of this has changed over time as the sport evolves with more leniency given to the extent of this time and more power to individual referees to give their own judgment, leading to conflicting decisions not only between different referees but even the same referee between points.

Our program strives to revolutionize fencing refereeing by allowing competition runners to access software that provides them with data-driven decisions by integrating and optimizing machine learning technology, as well as implementing an objective way to back them up under scrutiny from participants and fans alike.

This technology is applicable not only to fencing competitions but also when practicing [9]. Fencers are often unable to have a third person on the side refereeing them, and disagreements often occur without a referee, which reduces the effectiveness of practice and slows down the process.

Some AI fencing referees for foil and sabre have been proposed to resolve the issue of inconsistencies in referees, which allow the users to attain a more impartial judgment on the decision of a point. For example, AllezGo is a full neural network model-based referee that is trained on existing labeled videos in order to make a decision. It uses “Fencing-AI” by Douglas Sholto to obtain marked videos, which are then used to train his model and allow it to make calls [10]. However, neural networks are opaque in the sense that their specific parameters are not visible to humans and can only be seen in terms of their weight in the decision-making process. Consequently, their output cannot be explained, and the dataset on which the model is trained could lead to potential systematic unreliabilities and biases. Therefore, when the model’s decisions are questioned, the user cannot offer an answer to the reasoning behind the decision, which is often asked by fencers in competitions. Due to the high stakes of competitions, it can harm both the fairness and bring to question the results of the tournament if the model is used to judge it, especially in more controversial cases. There is another AI-based fencing referee that is dealing with sabre fencing exclusively, “How to use convolutional neural networks to build a referee” by Rui Guo [7]. The product of the aforementioned paper is similar to AllezGo in its method of obtaining a decision through a trained neural model with the ability to detect blade contacts using a second neural network which serves to increase its accuracy in sabre fencing. This presents it the same problem as AllezGo as well as the inability to be applied to both foil and sabre because it will be more inaccurate in the context of foil.

In this paper, we follow the same line of research by trying to help the user find an unbiased and objective referee. Our method starts with preprocessing the video and marking it with keypoints, and then we analyze the video frame by frame to track certain critical statistics such as velocity, acceleration, and extension of the weapon arm to help us to make the decision. Compared to AllezGo and similar complete neural network-based referees, our parameters are completely transparent and open for the viewer to see and understand the logic behind why the decisions were made. This allows the tournament organizers to ensure that the decisions are completely unbiased and objective, as well as easily demonstrated to both competitors and sponsors. Furthermore, the ease of access and mutation of the parameters and thresholds allow us to quickly change and modify our referee to adapt to the ever-changing rules and preferences of the governing body without the need for an adaptation period where the model must be retrained which is both time-consuming and extremely expensive financially.

Another feature of our program is that it is easily extendable to sabre fencing as well. In sabre fencing, the concept of priority also exists to determine who receives the points, with differences in the specific rules that govern how movement affects priority. Using our program, we can

change it to fit sabre fencing and add other statistics that we would like to track without having to completely retrain our model.

The rest of the paper is organized as follows: Section 2 gives the details on the challenges that we met during the experiment and designing the sample; Section 3 focuses on the details of our solutions corresponding to the challenges that we mentioned in Section 2; Section 4 presents the relevant details about the experiment we did, following by presenting the related work in Section 5. Finally, Section 6 gives the conclusion remarks, as well as pointing out the future work of this project.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. Model Selection

The first challenge was the selection of a model that we would use to track the fencers and identify the key points and components. At first, we used a model called MediaPipe in combination with YoloV5 to complete that task. Unfortunately, that model was only capable of utilizing the CPU of the computer and, with a computation and graphical analysis heavy program like ours, was struggling even on very powerful CPUs as well as had tracking issues, sometimes even failing to detect one of the fencers [8]. This made it unable to accurately obtain the critical statistics and allow for it to be run on end users' portable laptops that are present at both competitions and in training sessions at the club. Since this model is a critical part of our project, we had to pick the one with the best balance of performance and quality of keypoint detection in order for our program to be practical.

2.2. Writing the rules/Defining thresholds

Another challenge that we faced was writing the rules based on the data that would ultimately make the decision. In fencing, there is no specific definition of what an attack is, which is what allows the fencer to have priority. The only defining characteristic is that the fencer must attack within "fencing time," however, this phrase has no definition either. The definition of an attack and priority has largely been decided through meetings of the International Fencing Federation which sets and changes the definition over time and creates new norms with how international referees call the actions, which then is eventually spread to the federations and competitions of each country. Since there is no technical definition for these terms, it is very difficult to translate the human notions of "moving first," "extending first," and "advancing" into mathematical formulas that take advantage of the computer's computational power and ability to analyze the bout in stop motion.

2.3. Lack of Datasets

Finally, a big obstacle in finding a solution to our problem is the lack of datasets which are representative of the different situations that occur on piste. We needed to find both general and edge cases to ensure that the program would work as intended no matter what happened in the bout, as every fencing bout is different, with each fencer having different styles and natural movements. We have to account for a great variety of speeds, directions, and ways fencers move their bodies and extend their arms. It takes not just time but also expert knowledge to be able to find and identify videos for each and every possible situation that could arise. Furthermore, accounting for lighting, camera quality, position, and framerate, as well as the number of people

in the frame, makes it extremely difficult to find high-quality videos which we can use the statistics from as the reference in order to create the basic rules and settle on rational thresholds that could work for a larger superset of bouts.

3. SOLUTION

AIFR is a novel knowledge-based AI-driven fencing judging solution that aims to fill the niche of being a customizable and transparent application that will allow tournament organizers to build trust with their competitors and the people providing funding for the event by being able to definitively say that their judging is objective and unbiased. It is composed of two sub-programs, which are the processing component and the player component, which are loosely coupled in order to ensure our ability to evolve our design to other potential software architectures, such as client-server if we wanted to potentially create a web endpoint or a phone application due to the inherent limitations of those two technologies. The processing app utilizes Yolov7's keypoint detection capabilities in order to sift through a video frame by frame while tracking statistics which are easy to create and tinker with to achieve the exact specifications for the current tournament and refereeing convention [8]. Then, the player app is able to play back the video either frame-by-frame or in full motion with the key points and statistics displaying every moment's data in an easily readable format that contains graphs and flags which show the current state of the bout with the current threshold settings. We have also implemented a way to cache the results of previous operations in order to minimize time spent verifying outputs and also worked on optimizing its performance so that the app can function even on less-than-stellar machines, while still retaining most of its accuracy and producing the result within a reasonable time frame.

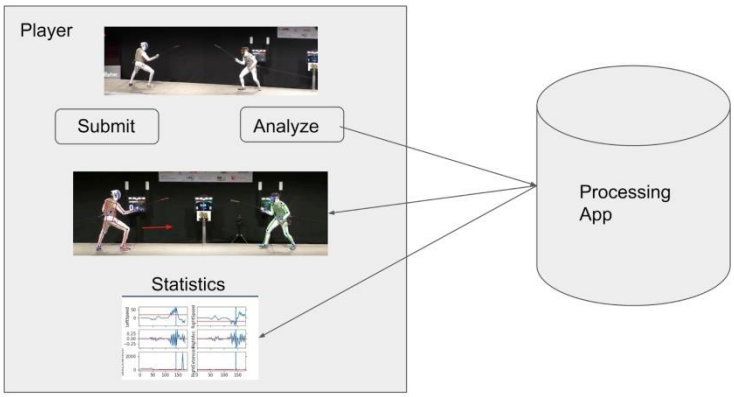


Figure 1. Overview of the solution

```
while (cap.isOpened):
    # Capture each frame of the video.
    ret, frame = cap.read()
    if ret:
        orig_image = frame
        image = cv2.cvtColor(orig_image, cv2.COLOR_BGR2RGB)
        image = letterbox(image, (frame_width), stride=64, auto=True)[0]
        image_ = image.copy()
        image = transforms.ToTensor()(image)
        image = torch.tensor(np.array([image.numpy()]))
        image = image.to(device)
        image = image.half()

        # Get the start time.
        start_time = time.time()
        with torch.no_grad():
            output, _ = model(image)
        # Get the end time.
        end_time = time.time()
        # Get the fps.
        fps = 1 / (end_time - start_time)
        # Add fps to total fps.
        total_fps += fps
        frameCheck = 3
        if frame_count % frameCheck == 0 and frame_count >= frameCheck:
            leftFencer.updateSpeed(frameCheck)
            leftSpeed.update(frame_count, leftFencer.speed)
            leftAcceleration.update(frame_count, leftFencer.acceleration)
            leftExtension.update(frame_count, leftFencer.getExtension())
            print("Left speed:", leftFencer.speed)
            rightFencer.updateSpeed(frameCheck)
            rightSpeed.update(frame_count, rightFencer.speed)
            rightAcceleration.update(frame_count, rightFencer.acceleration)
            rightExtension.update(frame_count, rightFencer.getExtension())
            print("Right speed:", rightFencer.speed)
```

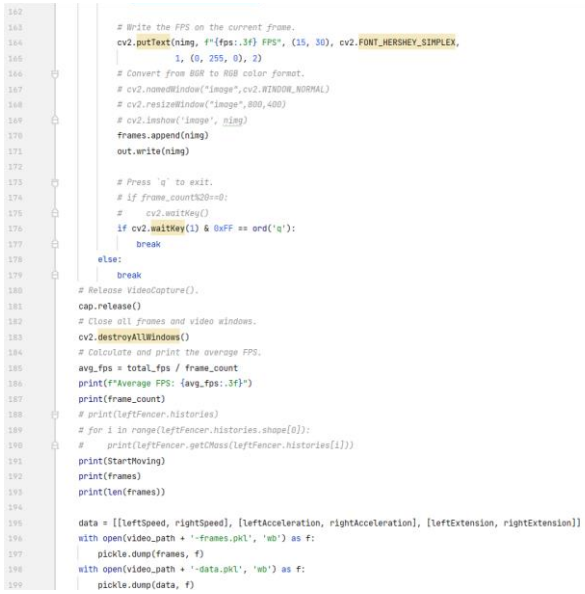
Figure 2. Screenshot of code 1

This code is the main processing loop of the program that goes through the video frame by frame and tracks the keypoints as well as records the required statistics to be sent to the player to be analyzed. It also draws the keypoints and lines on the limbs of the fencers. In lines 93-100, we convert the video frame by frame into the appropriate color and format for Yolov7 and then send it to the memory of the device that is used for the model to analyze. Then in lines 103-107 we start to track the time it takes for the model to process the image and feed the image through it. In lines 112-123, we update the key statistics using functions we have written every three frames to smooth out the data and minimize noise.

```
    # Increment frame count.
    frame_count += 1
    window.write_event_value("Update Text", "")
    output = non_max_suppression(output, 0.25, 0.65, nc=model.yaml['nc'], nkpt=model.yaml['nkpt'],
                                 kpt_label=True)
    output = output_to_keypoint(output)
    nimg = image[0].permute(1, 2, 0) * 255
    nimg = nimg.cpu().numpy().astype(np.uint8)
    nimg = cv2.cvtColor(nimg, cv2.COLOR_RGB2BGR)
    leftIndex = 0
    if output.shape[0] == 2:
        leftCenter = rules.getCMass(rules.getPosition(output[0, 7:].T))
        rightCenter = rules.getCMass(rules.getPosition(output[1, 7:].T))
        print("leftCenter", leftCenter, "rightCenter", rightCenter)
        if (leftCenter[0] > rightCenter[0]):
            leftIndex = 1

    for idx in range(output.shape[0]):
        plot_skeleton_kpts(nimg, output[idx, 7:].T, 3)

        print("left index", leftIndex, "right index", 1 - leftIndex)
        if idx == leftIndex:
            rules.processKeyPoints(output[idx, 7:].T, 3, leftFencer)
        if idx == 1 - leftIndex:
            rules.processKeyPoints(output[idx, 7:].T, 3, rightFencer)

        # Comment/Uncomment the following lines to show bounding boxes around persons.
        xmin, ymin = (output[idx, 2] - output[idx, 4] / 2), (output[idx, 3] - output[idx, 5] / 2)
        xmax, ymax = (output[idx, 2] + output[idx, 4] / 2), (output[idx, 3] + output[idx, 5] / 2)
        cv2.rectangle(
            nimg,
            (int(xmin), int(ymin)),
            (int(xmax), int(ymax)),
            color=(255, 0, 0),
            thickness=1,
            lineType=cv2.LINE_AA
        )
```

Figure 3. Screenshot of code 2

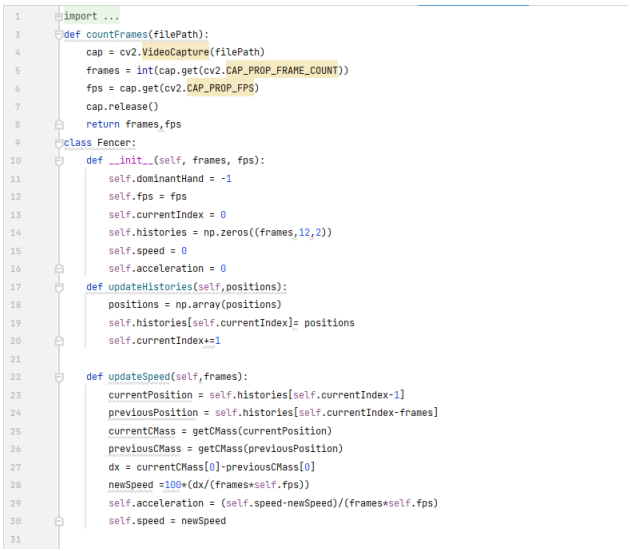
The next step we take is in 128-134, here we convert the image back to the CV2 format. From lines 135 through 140 we find the center of mass of each fencer to be used in identifying whether each fencer is on the left or the right. In lines 142-160, the detections are processed using the processKeyPoints() method of the rules module, and the keypoints are drawn on the image.



```
            # Write the FPS on the current frame.
            cv2.putText(nimg, f"{fps:.3f} FPS", (15, 30), cv2.FONT_HERSHEY_SIMPLEX,
                        1, (0, 255, 0), 2)
            # Convert from BGR to RGB color format.
            # cv2.namedWindow("image",cv2.WINDOW_NORMAL)
            # cv2.resizeWindow("image",800,400)
            # cv2.imshow('image', nimg)
            frames.append(nimg)
            out.write(nimg)

            # Press `q` to exit.
            # if frame_count%20==0:
            #     cv2.waitKey()
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
        else:
            break
    # Release VideoCapture().
    cap.release()
    # Close all frames and video windows.
    cv2.destroyAllWindows()
    # Calculate and print the average FPS.
    avg_fps = total_fps / frame_count
    print(f"Average FPS: {avg_fps:.3f}")
    print(frame_count)
    # print(leftFencer.histories)
    # for i in range(leftFencer.histories.shape[0]):
    #     print(leftFencer.getCMass(leftFencer.histories[i]))
    print(StartMoving)
    print(frames)
    print(len(frames))

    data = [[leftSpeed, rightSpeed], [leftAcceleration, rightAcceleration], [leftExtension, rightExtension]]
    with open(video_path + '-frames.pkl', 'wb') as f:
        pickle.dump(frames, f)
    with open(video_path + '-data.pkl', 'wb') as f:
        pickle.dump(data, f)
```

Figure 4. Screenshot of code 3

Lastly, we write down the frames per second on the image and add it to the list of images that have already been processed.



```
import ...
def countFrames(filePath):
    cap = cv2.VideoCapture(filePath)
    frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    fps = cap.get(cv2.CAP_PROP_FPS)
    cap.release()
    return frames,fps
class Fencer:
    def __init__(self, frames, fps):
        self.dominantHand = -1
        self.fps = fps
        self.currentIndex = 0
        self.histories = np.zeros((frames,12,2))
        self.speed = 0
        self.acceleration = 0
    def updateHistories(self,positions):
        positions = np.array(positions)
        self.histories[self.currentIndex]= positions
        self.currentIndex+=1

    def updateSpeed(self,frames):
        currentPosition = self.histories[self.currentIndex-1]
        previousPosition = self.histories[self.currentIndex-frames]
        currentCMass = getCMass(currentPosition)
        previousCMass = getCMass(previousPosition)
        dx = currentCMass[0]-previousCMass[0]
        newSpeed =100*(dx/(frames*self.fps))
        self.acceleration = (self.speed-newSpeed)/(frames*self.fps)
        self.speed = newSpeed

```

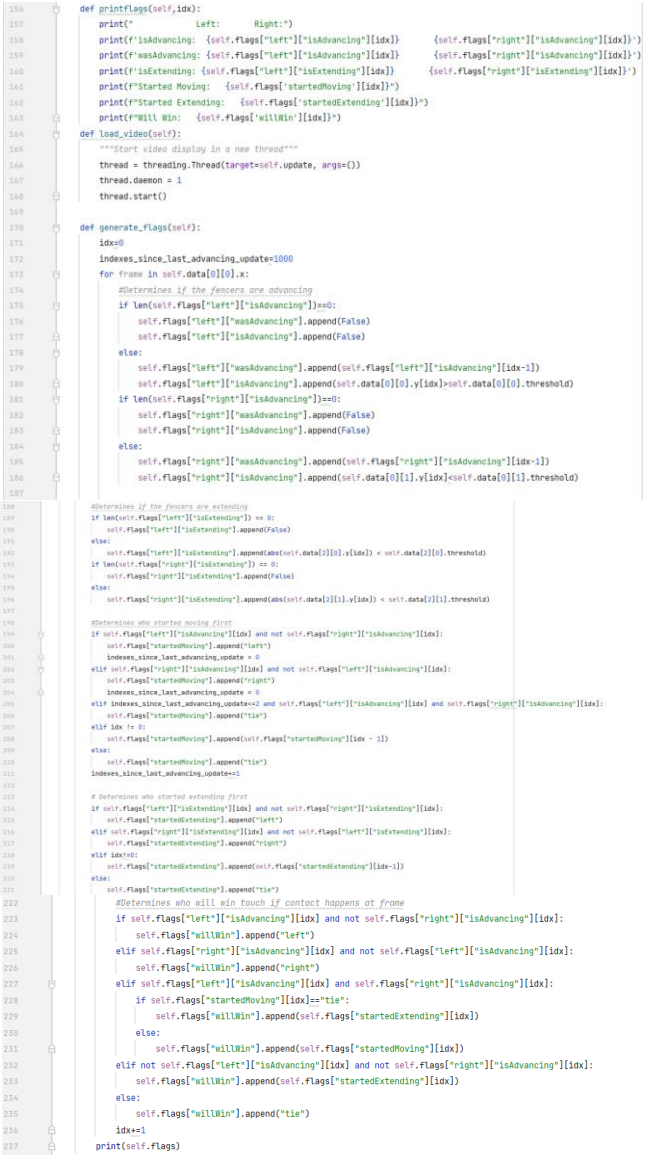
Figure 5. Screenshot of code 4

The function countFrames() counts the number of frames in the entire video. The __init__() method is the special dunder method which initializes all of the variables required for the object to function. The updateHistories() method updates the Histories two-dimensional array with the current positions of the fencer and updateSpeed() uses the position of the center of mass and the frames elapsed to calculate the speed and acceleration of the fencer with regards to the time passed.

```
    def getExtension(self):
        position = self.histories[self.currentIndex-1]
        if (self.dominantHand == -1):
            shoulder = position[1]
            elbow = position[3]
            hand = position[5]

        elif (self.dominantHand == 1):
            shoulder = position[0]
            elbow = position[2]
            hand = position[4]
            print("hi")
            print(elbow,shoulder, hand)
        slope = (hand[1]-shoulder[1])/(hand[0]-shoulder[0])
        intercept = shoulder[1]-slope*shoulder[0]
        distance = abs((slope*elbow[0] + intercept) - elbow[1])
        print("distance", distance)
        return distance
def getCMass(position):
    xPoint = ((((position[0][0]+position[1][0])/2) + ((position[6][0]+position[7][0])/2)))/2
    yPoint = ((((position[0][1] + position[1][1]) / 2) + ((position[6][1] + position[7][1]) / 2))) / 2
    return (xPoint, yPoint)
def processKeyPoints(kpts, steps,fencer):
    #Plot the skeleton and keypointsfor coco datatset
    num_kpts = len(kpts) // steps
    position = []
    for kid in range(5,num_kpts):
        x_coord, y_coord = kpts[steps * kid], kpts[steps * kid + 1]
        position.append([x_coord, y_coord])
    fencer.updateHistories(position)

def getPosition (inputArray):
    number = len(inputArray)//3
    position = []
    for kid in range(5, number):
        x_coord, y_coord = inputArray[3 * kid], inputArray[3 * kid + 1]
        position.append([x_coord, y_coord])
    return position
```

Figure 6. Screenshot of code 5

The method getExtension() identifies how extended a fencer's arm is with a numeric value by calculating the distance between the elbow and the line between the shoulder and hand through the keypoints found in the main processing loop. The getCMass() method finds the center of mass, processKeyPoints() transforms the keypints into a list of positions, and getPosition() is used generally to find keypoints of any detection and not just the fencers. These three methods offer crucial data that is then used in other methods and the actual rules to make judging decisions.



```
def printflags(self,idx):
    print("             Left:       Right:")
    print(f'isAdvancing:  {self.flags["left"]["isAdvancing"][idx]}       {self.flags["right"]["isAdvancing"][idx]}')
    print(f'wasAdvancing: {self.flags["left"]["isAdvancing"][idx]}       {self.flags["right"]["isAdvancing"][idx]}')
    print(f'isExtending: {self.flags["left"]["isExtending"][idx]}       {self.flags["right"]["isExtending"][idx]}')
    print(f"Started Moving:   {self.flags['startedMoving'][idx]}")
    print(f"Started Extending:   {self.flags['startedExtending'][idx]}")
    print(f"Will Win:   {self.flags['willWin'][idx]}")
def load_video(self):
    """Start video display in a new thread"""
    thread = threading.Thread(target=self.update, args=())
    thread.daemon = 1
    thread.start()

def generate_flags(self):
    idx=0
    indexes_since_last_advancing_update=1000
    for frame in self.data[0][0].x:
        #Determines if the fencers are advancing
        if len(self.flags["left"]["isAdvancing"])==0:
            self.flags["left"]["wasAdvancing"].append(False)
            self.flags["left"]["isAdvancing"].append(False)
        else:
            self.flags["left"]["wasAdvancing"].append(self.flags["left"]["isAdvancing"][idx-1])
            self.flags["left"]["isAdvancing"].append(self.data[0][0].y[idx]>self.data[0][0].threshold)
        if len(self.flags["right"]["isAdvancing"])==0:
            self.flags["right"]["wasAdvancing"].append(False)
            self.flags["right"]["isAdvancing"].append(False)
        else:
            self.flags["right"]["wasAdvancing"].append(self.flags["right"]["isAdvancing"][idx-1])
            self.flags["right"]["isAdvancing"].append(self.data[0][1].y[idx]<self.data[0][1].threshold)

    #Determines if the fencers are extending
    if len(self.flags["left"]["isExtending"]) == 0:
        self.flags["left"]["isExtending"].append(False)
    else:
        self.flags["left"]["isExtending"].append(abs(self.data[2][0].y[idx]) < self.data[2][0].threshold)
    if len(self.flags["right"]["isExtending"]) == 0:
        self.flags["right"]["isExtending"].append(False)
    else:
        self.flags["right"]["isExtending"].append(abs(self.data[2][1].y[idx]) < self.data[2][1].threshold)

    #Determines who started moving first
    if self.flags["left"]["isAdvancing"][idx] and not self.flags["right"]["isAdvancing"][idx]:
        self.flags["startedMoving"].append("left")
        indexes_since_last_advancing_update = 0
    elif self.flags["right"]["isAdvancing"][idx] and not self.flags["left"]["isAdvancing"][idx]:
        self.flags["startedMoving"].append("right")
        indexes_since_last_advancing_update = 0
    elif indexes_since_last_advancing_update<=2 and self.flags["left"]["isAdvancing"][idx] and self.flags["right"]["isAdvancing"][idx]:
        self.flags["startedMoving"].append("tie")
    elif idx != 0:
        self.flags["startedMoving"].append(self.flags["startedMoving"][idx - 1])
    else:
        self.flags["startedMoving"].append("tie")
    indexes_since_last_advancing_update+=1

    # Determines who started extending first
    if self.flags["left"]["isExtending"][idx] and not self.flags["right"]["isExtending"][idx]:
        self.flags["startedExtending"].append("left")
    elif self.flags["right"]["isExtending"][idx] and not self.flags["left"]["isExtending"][idx]:
        self.flags["startedExtending"].append("right")
    elif idx!=0:
        self.flags["startedExtending"].append(self.flags["startedExtending"][idx-1])
    else:
        self.flags["startedExtending"].append("tie")
        #Determines who will win touch if contact happens at frame
        if self.flags["left"]["isAdvancing"][idx] and not self.flags["right"]["isAdvancing"][idx]:
            self.flags["willWin"].append("left")
        elif self.flags["right"]["isAdvancing"][idx] and not self.flags["left"]["isAdvancing"][idx]:
            self.flags["willWin"].append("right")
        elif self.flags["left"]["isAdvancing"][idx] and self.flags["right"]["isAdvancing"][idx]:
            if self.flags["startedMoving"][idx]=="tie":
                self.flags["willWin"].append(self.flags["startedExtending"][idx])
            else:
                self.flags["willWin"].append(self.flags["startedMoving"][idx])
        elif not self.flags["left"]["isAdvancing"][idx] and not self.flags["right"]["isAdvancing"][idx]:
            self.flags["willWin"].append(self.flags["startedExtending"][idx])
        else:
            self.flags["willWin"].append("tie")
        idx+=1
    print(self.flags)
```

Figure 7. Screenshot of code 6

This is the logic function of the player where the actual decisions are made using the methods where the flags are the indicator of critical data. The flags are printed with the printFlags() method and the multi-threading is done with the load_video() method to reduce the playback time. Finally, in the generateFlags() method we use multiple if and else if statements to check what each fencer is doing and make the final decision.

4. EXPERIMENT

4.1. Experiment 1

The experiment 1 will recruit experienced foil fencers and record their movements and actions during at least 100 bouts. The equipment used will be high-quality video cameras and a pose estimation system to accurately track the fencers' movements. The recorded data will then be analyzed, and the accuracy of the pose estimation system compared with the referee's decision-

making using statistical methods. The level of agreement between the referee's decision and the pose estimation system will be calculated and presented in a clear and concise manner, along with graphs. Overall, this experiment seeks to provide objective metrics and evidence-based decision-making tools that can be used to support referees in making more accurate and consistent decisions during fencing bouts.

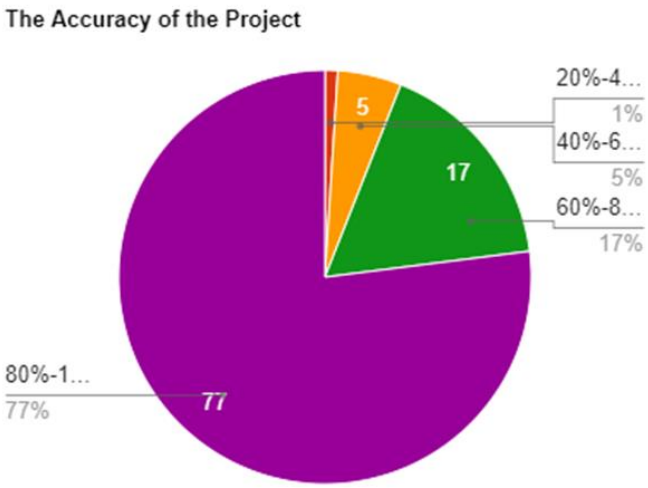


Figure 8. The accuracy of the project

Based on the data, it appears that the pose estimation system achieved a high level of accuracy in the majority of bouts. Specifically, it achieved an accuracy of 80%-100% in 77 out of the 100 bouts recorded. This suggests that the system was highly effective in accurately identifying and tracking the fencers' movements and actions.

It is also notable that the system achieved an accuracy of 60%-80% in 17 bouts. While this is a lower level of accuracy compared to the higher range, it is still a relatively good level of accuracy and suggests that the system was able to identify the majority of the fencers' movements and actions correctly.

However, the system's accuracy was lower in a smaller number of bouts, achieving an accuracy of 40%-60% in five bouts and 20%-40% accuracy in just one bout. While these lower levels of accuracy could be attributed to various factors, such as technical issues or complex movements by the fencers, it is still important to investigate and identify the causes to further improve the system's accuracy.

Overall, the data suggest that the pose estimation system is a promising technology for improving the consistency and accuracy of referee decision-making in foil fencing.

4.2. Experiment 2

The experiment 2 aims to collect user feedback from experienced foil fencers who have used the pose-estimation-based fencing refereeing system in previous competitions. The experiment involves conducting a survey to collect feedback from 100 participants. The survey will include questions on ease of use, accuracy, and preference for the pose-estimation-based system over traditional refereeing methods. The survey responses will be collected in score numbers from 0-10. Numbers will be analyzed to identify common themes and suggestions for improvement. The

findings from the experiment will be used to inform future developments and improvements to the system to enhance user satisfaction and effectiveness.

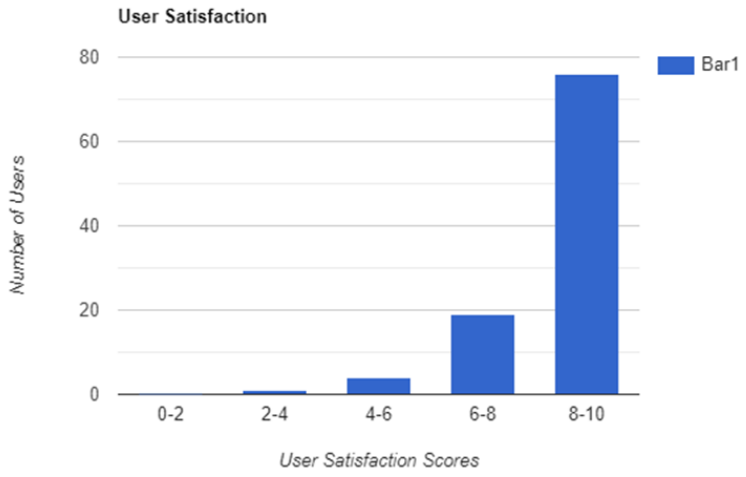


Figure 9. User Satisfaction

Based on the data results, the majority of experienced foil fencers who have used the pose-estimation-based fencing refereeing system in previous competitions rate the system highly. Specifically, 76 users rate the system as 8-10 out of 10, indicating a high level of satisfaction and effectiveness. Additionally, 19 users rate the system between 6-8, indicating that they found the system moderately effective. However, there were some users who gave lower ratings, with one user rating the system between 2-4 and no users rating the system as 0-2. Overall, the data suggests that the pose-estimation-based fencing refereeing system is well-received by experienced foil fencers and is an effective alternative to traditional refereeing methods.

The experiment 1 results indicate that the pose-estimation-based fencing refereeing system effectively addresses the challenge of inconsistent referee decision-making in foil fencing by achieving a high level of accuracy. The system achieved 80%-100% accuracy in the majority of bouts, but lower accuracy levels were observed in some bouts. The data suggest that the system is a promising technology for improving referee decision-making consistency and accuracy in foil fencing, but further investigation is needed to address technical issues and complex movements that may impact the system's performance. Overall, the experiment 1 results meet our expectations and provide valuable insights for future improvements.

The experiment 2 results demonstrate that the pose-estimation-based fencing refereeing system effectively addresses the challenge of inconsistent referee decision making in foil fencing, and is well-received by experienced foil fencers. The majority of users rated the system highly, indicating a high level of satisfaction and effectiveness. However, some users gave lower ratings, providing feedback for future improvements. Overall, the experiment 2 results meet expectations and provide valuable insights for enhancing user satisfaction and effectiveness.

5. RELATED WORK

This is a similar paper that proposes another solution to the problem through fully training an AI algorithm to examine and produce a refereeing decision specific to sabre fencing [11]. Compared to our application, this is able to be more accurate than ours in sabre fencing because it is able to detect blade actions. However, due to the fact that the process of training the model is both time-

consuming and expensive, it is hard to adapt it when the trends and rules of fencing change. Furthermore, because it is an AI algorithm it is extremely difficult to transparently understand the parameters which the algorithm is using to make a decision, therefore making it impossible to back up the decisions and prove that they are completely objective.

This application is also similar to ours and the previous one mentioned by being driven completely by a trained AI algorithm and is able to determine who had priority at any time and display it on screen for easy viewing [12]. When it is compared to our's, similarly it has the advantage of accuracy but it is more computation-heavy. The decisions cannot be made in a timely manner therefore, this application can only be used to overlay priority on the screen for people who don't know the rules to easily be able to understand what is happening.

YoloV7 was used in this projection as our main pose estimation algorithm [13]. It is also capable of object detection and instance segmentation which can be used to further improve our application. During our testing, we used an earlier model called YoloV5 in conjunction with MediaPipe to estimate the poses, however, this was slow and inaccurate. Switching to YoloV7, we were able to only use one algorithm which sped up the process and produced better performance. This algorithm was more complex compared to MediaPipe, and required us to do a more advanced setup for calculating the keypoints and critical statistics.

6. CONCLUSIONS

We first process and analyze the videos to obtain the keypoints and limbs of the fencers [14]. Then, we use these keypoints to calculate the various data that will be used in determining the priority. And finally, the data is used by the rules to produce the final decision. We also tested the project with two detail designed experiments.

The experiment 1 results demonstrate that the pose-estimation-based fencing refereeing system effectively addresses the challenge of inconsistent referee decision-making in foil fencing by achieving a high level of accuracy in the majority of bouts. However, further investigation is needed to improve the system's accuracy in the cases where lower accuracy levels were achieved.

The experiment 2 results provide insight into areas for improvement. While the majority of users rated the system highly, there were some who gave lower ratings. By analyzing the survey responses and identifying common themes and suggestions for improvement, future developments and improvements can be made to enhance user satisfaction and effectiveness even further.

The first limitation we have is the lack of blade detection [15]. Even though most points don't result from blade actions, it is necessary to have it if there is no presence of a human referee, The second limitation is that there must be changes and tweaks to the specific thresholds of our algorithm due to the difference of camera position and quality at each competition or venue. The third limitation is that we cannot have a third person in the frame because our pose detection works by finding a left and right fencer to determine the data, so in the presence of a third person, the algorithm may not be able to determine who the fencers are.

To counter the limitations, we could train a custom model to detect the blades and therefore be able to track them and include blade actions when making the decision. Similarly, we can train a model to detect the fencers based on their masks and also an automatic threshold-determining feature that requires the user to input the position of the camera and its specifications.

REFERENCES

[1] Witkowski, Mateusz, et al. "Visual perception strategies of foil fencers facing right-versus left-handed opponents." Perceptual and Motor Skills 125.3 (2018): 612-625.

[2] Roi, Giulio S., and Diana Bianchedi. "The science of fencing: implications for performance and injury prevention." Sports medicine 38 (2008): 465-481..

[3] Wu, Wentong, et al. "Application of local fully Convolutional Neural Network combined with YOLO v5 algorithm in small target detection of remote sensing image." PloS one 16.10 (2021): e0259283.

[4] Lugaresi, Camillo, et al. "Mediapipe: A framework for perceiving and processing reality." Third Workshop on Computer Vision for AR/VR at IEEE Computer Vision and Pattern Recognition (CVPR). Vol. 2019. 2019.

[5] Moore, Kevin C., Frances ME Chow, and John YH Chow. "Novel lunge biomechanics in modern Sabre fencing." Procedia engineering 112 (2015): 473-478.

[6] Roček, Michal. "Educational Activities of the International Fencing Federation (FIE)“FIE Budapest Coaching Academy”." Studia sportiva 13.2 (2019): 94-96.

[7] Abdel-Hamid, Ossama, et al. "Convolutional neural networks for speech recognition." IEEE/ACM Transactions on audio, speech, and language processing 22.10 (2014): 1533-1545.

[8] Wang, Chien-Yao, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors." arXiv preprint arXiv:2207.02696 (2022).

[9] Harmenberg, J., et al. "Comparison of different tests of fencing performance." International journal of sports medicine 12.06 (1991): 573-576.

[10] Hayward, Matt W., and Graham IH Kerley. "Fencing for conservation: restriction of evolutionary potential or a riposte to threatening processes?." Biological Conservation 142.1 (2009): 1-13..

[11] Segler, Marwin HS, Mike Preuss, and Mark P. Waller. "Planning chemical syntheses with deep neural networks and symbolic AI." Nature 555.7698 (2018): 604-610.

[12] Bohr, Adam, and Kaveh Memarzadeh. "The rise of artificial intelligence in healthcare applications." Artificial Intelligence in healthcare. Academic Press, 2020. 25-60.

[13] Jiang, Kailin, et al. "An Attention Mechanism-Improved YOLOv7 Object Detection Algorithm for Hemp Duck Count Estimation." Agriculture 12.10 (2022): 1659.

[14] Tsolakis, Charilaos, and George Vagenas. "Anthropometric, physiological and performance characteristics of elite and sub-elite fencers." Journal of human kinetics 23.1 (2010): 89-95.

[15] Du, Ying, et al. "Damage detection techniques for wind turbine blades: A review." Mechanical Systems and Signal Processing 141 (2020): 106445.