Exploring Computer Vision-Based AI-Assisted Coaching for Youth Football Players

Emil Folke Gustafsson
Abstract

Recently advances in computer vision have been made with the aid of artificial intelligence. This has made tracking sports in real-time a possibility. Specifically, this project aim to track objects in a football exercise in real-time with a single low-angle video camera, and quickly create feedback. Detecting the players during the exercise was mostly successful, but the ball could only be detected in certain areas of the playing area. Keeping track of the different players in the exercise proved to be difficult with the setup. Alternate ways to provide feedback without knowing the identities of the players were possible but limited. To be able to reliably provide insightful feedback the system would likely need to be changed to a high-angle or multi-camera system.
Acknowledgements

I would like to thank David Sumpter and all the students and supervisors who participated in our group supervision meetings for their valuable input. Thanks to Katie Winkle and the Social Robotics Lab for lending me the necessary equipment and their input. I would also like to thank Ida-Maria Sintorn for organising one of the data collection occasions.
Glossary of Key Terms

The Assignment Problem In this setting, finding the optimal way to assign detections to tracked objects so that the total cost is minimised.

Direct Linear Transform A method to estimate transformation parameters from observed point correspondences using matrix algebra and solving linear equations. Described further in B.1.

Homography Matrix Defines how points in one image correspond to points in another image when both images are of the same scene.

Hungarian Algorithm Algorithm used for solving the Assignment Problem. Described further in B.2.

Identity Switch When the identity of one object is incorrectly assigned to a different object.

Kalman Filtering An algorithm utilised in object tracking, which processes a sequence of measurements observed over time, including statistical noise and inaccuracies, to generate estimates of unknown variables that are generally more precise than those derived from individual measurements alone. Described further in B.3.

Pitch Control At a location is the probability that a team or player will gain possession if the ball moves to a location.
Contents

1 Introduction 2

2 Method 4
   2.1 Multiple Object Detection . . . . . . . . . . . . . . . . . . . . . . 4
      2.1.1 Data . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
      2.1.2 Model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
   2.2 Multiple Object Tracking . . . . . . . . . . . . . . . . . . . . . . 5
      2.2.1 DeepSORT . . . . . . . . . . . . . . . . . . . . . . . . . . 5
      2.2.2 ByteTrack . . . . . . . . . . . . . . . . . . . . . . . . . . 5
      2.2.3 Combining Lost Tracks . . . . . . . . . . . . . . . . . . . . 6
   2.3 Real-Time . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
      2.3.1 Pitch Control . . . . . . . . . . . . . . . . . . . . . . . . . 7
      2.3.2 Real-Time Application . . . . . . . . . . . . . . . . . . . . . 7

3 Results 8
   3.1 Multiple Object Detection . . . . . . . . . . . . . . . . . . . . . . 8
   3.2 Multiple Object Tracking . . . . . . . . . . . . . . . . . . . . . . 9
   3.3 Real-Time Application with Pitch Control . . . . . . . . . . . . . 10

4 Discussion 11
   4.1 Future Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

A YOLOv8 Model 15
   A.1 Figures from Training . . . . . . . . . . . . . . . . . . . . . . . . 15
   A.2 Model Architecture . . . . . . . . . . . . . . . . . . . . . . . . . . 17

B Key Terms 19
   B.1 Direct Linear Transform . . . . . . . . . . . . . . . . . . . . . . . 19
   B.2 The Hungarian Algorithm . . . . . . . . . . . . . . . . . . . . . . 20
   B.3 Kalman Filtering . . . . . . . . . . . . . . . . . . . . . . . . . . . 21

C Additional Figures 22
Populärvetenskaplig sammanfattning

Framsteg inom datorseende med hjälp av artificiell intelligens har öppnat nya möjligheter att spåra både spelare och boll under fotbollsövningar i realtid. I projektet undersöks möjligheterna att snabbt skapa ett system som kan göra just detta med redan existerande metoder och på så sätt också snabbt ta fram återkoppling till spelarna som utfört övningen. Projektet fokuserade på en specifik övning där två anfallande spelare försöker göra mål mot en ensam förvarare och en målvakt. En videokamera monterad på en tripod som resulterade i en kamerahöjd på 1,6 m användes för att filmma övningen. Varje bildruta från kameran behandlades i två steg. Först av en modell som detekterar objekt i bildrutan och klassificerar dem. Sedan skickas detektionerna till nästa steg där de matchas mot objekt från tidigare bildrutor baserat på position och utseende.

1 Introduction

The emergence of AI in recent years has spread over many fields, and sports, particularly football, is no exception. In football, AI is used or tested for several tasks, including predicting outcomes, aiding in sport medicine, refining tactics and improving recruitment. A prevalent discussion in the field revolves around the detection and tracking of players and the football. In the late 1990s and early 2000s, this was done using computer vision techniques such as blob detection and template matching. More recently, machine learning has generated interest as a potential tool, as indicated by numerous blog posts and academic papers. However, these discussions often focus on the challenges of analysing footage from sources, such as broadcast feeds and multi-camera setups.

This project seeks to investigate the potential and constraints of employing a low-angle single-camera setup with almost real-time feedback capabilities. The ultimate goal of such a system would be to provide quick feedback during youth team practice sessions. As this project is the first step towards this goal it will try to answer the following questions:

- Given the recent advances in AI, can they be used to quickly build a system which tracks players accurately?

- What is the feasibility of a tracking system? What can we expect in terms of real-time feedback?

The project focuses on a particular 2v1 exercise, where two attacking players attempt to score a goal against a single defender and a goalkeeper. An exercise like this was chosen because it involves a limited number of players within a manageable playing area.

For such a system to be able to provide almost real-time feedback, player and ball positions would need to be collected fully automatically. This is different from how much of football data is collected as it normally uses human input together with AI. Multiple object detection in computer vision involves the location of objects in images or video frames. Deep learning models for this task can be divided into two categories: one-stage and two-stage models. Two-stage models such as Faster R-CNN (Region-based Convolutional Neural Network) and Mask R-CNN first generate region proposals and then classify objects within those regions. One-stage models like YOLO (You Only Look Once) and its successors, on the other hand, use predetermined regions or directly predict the classes and bounding boxes of objects. One-stage models are typically faster, while two-stage models can be more accurate.
To ensure the association of player detections across frames as the same individual, a multiple-object tracker is needed. Occlusions and motion blur make this task a challenging one, potentially resulting in players being unrecognised or misidentified. Different tracking methods employ different strategies to address these challenges. ByteTrack and DeepSORT (Simple Online and Real-Time Tracking), the two methods selected, employ separate approaches. DeepSORT builds upon SORT. SORT uses Kalman filtering to predict the positions and velocities of tracked objects, and the Hungarian algorithm to match new detections with existing tracks. DeepSORT adds a CNN (Convolutional Neural Network) for re-identification to SORT, in order to improve robustness. ByteTrack utilises detections with low confidence scores that would typically be discarded, but keeping them separate from high-scored detections. Similarly to DeepSORT it uses Kalman filtering to predict the upcoming position of the tracked objects. It then associates the new detections in two stages, first with the high-score detections, and secondly the remaining unmatched tracks with the low-score detections. In this way ByteTrack fully utilises the output from the object detection model.
2 Method

This section outlines the methodology used for the project. The same equipment was used throughout the work. A Panasonic HC-V380 video camera, a tripod with a camera height of 1.6 m, a video capture device and a laptop.

2.1 Multiple Object Detection

This subsection describes how the data were used to train and evaluate the multiple object detection model used and how it was trained.

2.1.1 Data

Data for training and evaluating methods were collected on two separate occasions, all using the camera and tripod specified above. The first occasion involved recording 93 clips of university students engaging in the 2v1 exercise outlined in the Introduction. These clips vary in duration from four to 15 seconds and were captured from three camera positions. Additional images of three balls being kicked around the same football pitch used in the first occasion was collected to obtain more examples of footballs. On the second occasion, 18 clips of similar duration and featuring the same training exercise as on the first occasion were captured. This time with school children and every player wearing distinctly different coloured tops.

To facilitate result evaluation, clips from the first and third occasions were edited to ensure all four players and the ball remained within the frame in every frame, although occlusion by other objects may occur. To establish an annotated image dataset to train the object detection model, frames from the collected data were manually annotated. A total of 307 images were used for model training, 34 of them reserved for validation purposes. The validation set was only used as an early indicator of the result, as it is very small. Results from training were later evaluated together with the tracking methods.

2.1.2 Model

A pre-trained YOLOv8 model trained on COCO (Common Objects in Context) images was trained further on the dataset described above. The backbone of the pre-trained model was kept frozen during training for 15 epochs. The model has 3.2 million weights. Additional information about the training process can be found in Appendix.
2.2 Multiple Object Tracking

In this subsection the parameters used for the two tracking methods are presented, and the method for handling lost tracks is described.

2.2.1 DeepSORT

DeepSORT was implemented using the code associated with the article[32], with the assistance of another GitHub repository[6] to implement it faster. Minor changes were made to make the original provided by the paper code run in the desired environment and to make keeping track of classes of objects more convenient. The parameters used for the method were derived primarily from the default settings and some trial and error. The parameters are listed in table 1.

Table 1: Table containing DeepSORT parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tracking threshold</td>
<td>0.4</td>
</tr>
<tr>
<td>n initialisation</td>
<td>3</td>
</tr>
<tr>
<td>max age</td>
<td>15</td>
</tr>
<tr>
<td>max cosine distance</td>
<td>0.4</td>
</tr>
</tbody>
</table>

The tracking threshold is the minimum confidence score for a detection not to be discarded, initialisation number is the number of consecutive frames an object needs to be detected for a track to be created, maximum age is the number of consecutive frames an object can be missing before it is considered to have left the scene, the maximum cosine distance is the maximum cosine distance between feature vectors for a detection and track to be associated.

2.2.2 ByteTrack

ByteTrack was implemented using the code provided in the GitHub repository[34], along with helpful utilities from another GitHub repository[27]. Slight modifications were applied to enable the code from the paper to function in the desired environment and to make keeping track of classes of objects more convenient. The method’s parameters were predominantly based on default settings, with some adjustments through trial and error. The parameters are listed in Table 2. The matching threshold is the lower threshold for a detection to be associated to a track. Tracking threshold and maximum age are explained in the previous subsection.

Table 2: Table containing ByteTrack parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tracking threshold</td>
<td>0.3</td>
</tr>
<tr>
<td>max age</td>
<td>15</td>
</tr>
<tr>
<td>matching threshold</td>
<td>0.8</td>
</tr>
</tbody>
</table>
2.2.3 Combining Lost Tracks

Since there are only for players taking part in the exercises, lost tracks were combined until for unique players remained. This was done by removing very short tracks. Then the shortest remaining tracks was patched to a different track firstly based on overlapping in time and secondly by start and end position on pitch.

2.3 Real-Time

To enable any type of analysis of the results produced by object detection and tracking, the bounding boxes in the frames needed to be converted into 2d-position on the football pitch. This was achieved by marking four places in the field and defining the coordinates of them, as shown in Figure 1. Together with the pixel coordinates of these markings using the function \texttt{findHomography()} from the OpenCV library[3]. The function uses the direct linear transform to estimate the transformation matrix[21].

Using the equipment described in the beginning of this section, video frames are sent from the video camera to the laptop via a HDMI-cable and a video capture device. The object detection model then produces detections for the tracking method. Lastly the positions of the tracked objects are stored.

Figure 1: Figure showing the four marked positions in a video frame and diagram of the playing area from the first data collection occasion.
2.3.1 Pitch Control

Pitch control was implemented by adapting an existing implementation[26] following the methodology of Spearman[29]. To make individual pitch control possible, changes were made to the existing implementation. Since the calculation of the pitch control is done after a round is finished, detected position could be smoothed using both past and future positions. A rolling average with window size of ten frames was used. The same was done for the approximated velocities.

2.3.2 Real-Time Application

A windowed application in Python was developed but only tested on recorded videos using Tkinter[18] and OpenCV[3]. It is calibrated by marking the four corners of the playing area on the pitch and then the same points in an image of the playing area. The camera needs to be kept in a fixed position, if it moves, it must be re-calibrated. The rounds of the exercise would need to be started and ended manually. After a round, four annotated frames from the round are presented. After selecting a frame, the individual pitch control is calculated and displayed together with the frame.
3 Results

It was decided that annotating the ground truth for video clips would consume too much time. Therefore, the following metrics were used to evaluate object tracking: lost tracks, the number of unique player IDs exceeding four in a video clip, and identity switches, which were manually counted in the video clips after object detection, tracking, and combining additional track IDs. The average number of balls and players detected per frame was used to evaluate the performance of the object detection model. Ideally, this would be four players and one ball correctly detected in each frame. False detections were manually detected. In a limited number of clips from the second data collection occasion persons in the background were picked up. These video clips were excluded from the results in Section 3.2.

3.1 Multiple Object Detection

Initial tests from detecting objects in videos recorded at the first data collection occasion approximately eight meters away from playing area with pre-trained weights and image size 640x360 pixels detected very few balls. However, with custom trained weights and image size increased to 1024x576 the ball was detected in at least one frame in all clips. Figure 2 show where on the playing area the ball was detected, with the video camera positioned on three different positions on the right side of the area. Most of the ball detections were made on the same side as the camera. Player detections also suffered from occlusions, but not to the same extent.

Figure 2: Figure showing in which areas of the playing area the ball was detected. Video camera positioned on three positions on the right side of the figure.
3.2 Multiple Object Tracking

In order to get a quantitative measure of the robustness of the two tracking methods, the number of identity switches and lost tracks was counted in the 18 clips of the second occasion. The results are shown in Figure 3. DeepSORT had a lower number of lost tracks and identity switches, but is also computationally more expensive.

Figure 3: Comparison in number of lost tracks and identity switches in video clips. On these clips DeepSORT was the more robust tracking method compared to ByteTrack.
3.3 Real-Time Application with Pitch Control

Figure 4 shows a frame from the first data collection occasion with detections. In the following, the pitch control for both the team and individual players is displayed with estimated velocities. Teams were divided by visual inspection and not automatically. The timing of running object detection and storing results was performed on an AMD Ryzen 7 4800H, 2.90 GHz, 8-core CPU, and an NVIDIA GeForce GTX 1650 Ti, 4GB GPU. It ran at 30.7 fps on only the CPU and at 110 fps with the GPU. Additional images from the windowed application can be seen in Appendix C.

![Figure 4: Frame showing bounding boxes produced by object detection, and both team and individual pitch control.](image)
4 Discussion

As shown in Figure 2, occlusions and distance to the camera are problematic with a single camera setup. If the object detection was functioning as intended, the detections would more evenly distributed between the left and right sides of the playing area. It is important to note that the true ball positions are not equally distributed. For example, the low number of detections in the lower corners is probably partially explained by the ball not being there often. When each round starts, the players are spread out, which means the ball is only close to one player. Further in to the round, players are be more bunched together. Due to the lack of correct camera calibration during the second data collection session, the training images ended up comprising only a small portion of the videos used for Figure 2. Nonetheless, Figure 2 primarily illustrates areas on the playing field where balls were not detected. Therefore, these results are still deemed of interest.

Tracking proved to be difficult. An identity switch, when a player is reidentified incorrectly, in a round would make data from the round impossible to use and occurred in half of video clips as in Figure 3 with the more robust DeepSORT. It is not possible to know if an identity switch occurred without human review of the result. Different coloured tops were not enough to distinguish players consistently enough for this case. A less uniform background compare to videos from a broadcast feed could be a reason for objects not being reidentified.

As a result of the identities of the players not being accurately distinguished, a way of providing feedback with only detections was proposed. One limitation of the pitch control images is that they could potentially need to be interpreted by a coach. Timings of running the object detection model on recorded video clips show that only a CPU is likely only sufficient to run object detection in real-time, as if the frame rate drops too far below 30 fps, it would affect the accuracy of tracking methods.

4.1 Future Work

The results of this project show that the single low-angle camera setup has clear limitations. Occlusions hindering objects in crowded areas from being detected, and then the risk of it being misidentified when it returns in view of the camera. Since playing area is not so large and system not intended for fully grown individuals, it is possible that higher camera angle can have impact on identity switches. If not, an additional camera would be needed. To enable more feedback actions such as shots, passes, and dribbles would need to be recognised. With improved ball detections it could potentially be done by comparing the distance between the ball the player in possession. Recognising actions would enable evaluation of the riskiness of a pass or the quality of shooting opportunity.
References


A YOLOv8 Model

A.1 Figures from Training

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Epochs</td>
<td>15</td>
</tr>
<tr>
<td>Optimiser</td>
<td>AdamW</td>
</tr>
<tr>
<td>Batch size</td>
<td>16</td>
</tr>
<tr>
<td>Weight decay</td>
<td>0.0005 height</td>
</tr>
</tbody>
</table>

Figure 5: Distribution of classes in the created dataset.
Figure 6: Resulting confusion matrix from training.
A.2 Model Architecture

- **Layer 0**: Conv2d(3, 16, 3x3, S2, P1) → BN(16) → SiLU
- **Layer 1**: Conv2d(16, 32, 3x3, S2, P1) → BN(32) → SiLU
- **Layer 2**:
  - Conv2d(32, 32, 1x1) → BN(32) → SiLU
  - Conv2d(48, 32, 1x1) → BN(32) → SiLU
  - Bottleneck:
    * Conv2d(16, 16, 3x3, P1) → BN(16) → SiLU
    * Conv2d(16, 16, 3x3, P1) → BN(16) → SiLU
- **Layer 3**: Conv2d(32, 64, 3x3, S2, P1) → BN(64) → SiLU
- **Layer 4**:
  - Conv2d(64, 64, 1x1) → BN(64) → SiLU
  - Conv2d(128, 64, 1x1) → BN(64) → SiLU
  - Bottleneck (x2):
    * Conv2d(32, 32, 3x3, P1) → BN(32) → SiLU
    * Conv2d(32, 32, 3x3, P1) → BN(32) → SiLU
- **Layer 5**: Conv2d(64, 128, 3x3, S2, P1) → BN(128) → SiLU
- **Layer 6**:
  - Conv2d(128, 128, 1x1) → BN(128) → SiLU
  - Conv2d(256, 128, 1x1) → BN(128) → SiLU
  - Bottleneck (x2):
    * Conv2d(64, 64, 3x3, P1) → BN(64) → SiLU
    * Conv2d(64, 64, 3x3, P1) → BN(64) → SiLU
- **Layer 7**: Conv2d(128, 256, 3x3, S2, P1) → BN(256) → SiLU
- **Layer 8**:
  - Conv2d(256, 256, 1x1) → BN(256) → SiLU
  - Conv2d(384, 256, 1x1) → BN(256) → SiLU
  - Bottleneck:
    * Conv2d(128, 128, 3x3, P1) → BN(128) → SiLU
    * Conv2d(128, 128, 3x3, P1) → BN(128) → SiLU
- **Layer 9**: SPPF:
  - Conv2d(256, 128, 1x1) → BN(128) → SiLU
  - Conv2d(512, 256, 1x1) → BN(256) → SiLU
  - MaxPool2d(5, S1, P2)

17
• Layer 10: Upsample(SF2, 'nearest')
• Layer 11: Concat()
• Layer 12:
  - Conv2d(384, 128, 1x1) → BN(128) → SiLU
  - Conv2d(192, 128, 1x1) → BN(128) → SiLU
  - Bottleneck:
    * Conv2d(64, 64, 3x3, P1) → BN(64) → SiLU
    * Conv2d(64, 64, 3x3, P1) → BN(64) → SiLU
• Layer 13: Upsample(SF2, 'nearest')
• Layer 14: Concat()
• Layer 15:
  - Conv2d(192, 64, 1x1) → BN(64) → SiLU
  - Conv2d(96, 64, 1x1) → BN(64) → SiLU
  - Bottleneck:
    * Conv2d(32, 32, 3x3, P1) → BN(32) → SiLU
    * Conv2d(32, 32, 3x3, P1) → BN(32) → SiLU
• Layer 16: Conv2d(64, 64, 3x3, S2, P1) → BN(64) → SiLU
• Layer 17: Concat()
• Layer 18:
  - Conv2d(192, 128, 1x1) → BN(128) → SiLU
  - Conv2d(192, 128, 1x1) → BN(128) → SiLU
  - Bottleneck:
    * Conv2d(64, 64, 3x3, P1) → BN(64) → SiLU
    * Conv2d(64, 64, 3x3, P1) → BN(64) → SiLU
• Layer 19: Conv2d(128, 128, 3x3, S2, P1) → BN(128) → SiLU
• Layer 20: Concat()
• Layer 21:
  - Conv2d(384, 256, 1x1) → BN(256) → SiLU
  - Conv2d(384, 256, 1x1) → BN(256) → SiLU
  - Bottleneck:
    * Conv2d(128, 128, 3x3, P1) → BN(128) → SiLU
    * Conv2d(128, 128, 3x3, P1) → BN(128) → SiLU
• Layer 22: Detection Layer:
  - Conv and Conv layers → BN and SiLU activations.
  - DFL: Conv2d(16, 1, 1x1)
B Key Terms

B.1 Direct Linear Transform

The Direct Linear Transform method computes the homography matrix $H$, which describes the transformation between two planes.

1. Identify at least 4 corresponding points between two images or planes.

2. Represent points in homogeneous coordinates.

3. For each pair of corresponding points $(x_i, y_i, 1)$ and $(x'_i, y'_i, 1)$, construct linear equations that relate the points using the homography matrix $H$:

   \[
   x'_i \approx \frac{h_{11}x_i + h_{12}y_i + h_{13}}{h_{31}x_i + h_{32}y_i + h_{33}}, \quad y'_i \approx \frac{h_{21}x_i + h_{22}y_i + h_{23}}{h_{31}x_i + h_{32}y_i + h_{33}}
   \]

4. Combine these equations into a matrix equation $Ah = 0$, where $A$ is constructed from the coefficients:

   \[
   A = \begin{bmatrix}
   -x_i & -y_i & -1 & 0 & 0 & 0 & x_i x'_i & y_i y'_i & x'_i \\
   0 & 0 & 0 & -x_i & -y_i & -1 & x_i y'_i & y_i y'_i & y'_i
   \end{bmatrix}
   \]

5. Construct a matrix equation $Ah = 0$ where $A$ encodes linear equations relating corresponding points to $H$.

6. Use singular value decomposition to find the null vector $h$ of $A$.

7. Normalise $h$ to obtain $H$, ensuring $h_{33} = 1$.

8. Use least squares to improve $H$. 
B.2 The Hungarian Algorithm

The Hungarian algorithm is an optimisation algorithm used to solve the assignment problem. It can be explained by the following steps:

1. Calculate the cost matrix by using, e.g. cosine distance between feature vectors.

2. Find optimal assignment.
   - Subtract every element in each row by the lowest value in its row.
   - Perform the previous step but for columns.
   - Cover all zeros in the cost matrix with as few vertical or horizontal lines as possible.
   - If the number of lines is equal to the number of rows (or columns) the optimal assignment can be selected. Otherwise subtract the smallest uncovered value from all other uncovered elements and add it to elements covered twice. Repeat.
   - Optimal assignment is one zero for each column and row.
B.3 Kalman Filtering

State vector, $x$ and $y$ is the location of the pixel, $s$ the size, and $r$ the aspect ratio.

$$
\mathbf{x}_k = \begin{bmatrix}
x_k \\
y_k \\
s_k \\
r_k \\
\dot{x}_k \\
\dot{y}_k \\
\dot{s}_k 
\end{bmatrix}
$$

Predict

Predict state, where $A$ is the state-transition model.

$$
x_{k|k-1} = A x_{k-1}
$$

Predict covariance, where $Q$ is the covariance of the process noise.

$$
P_{k|k-1} = AP_{k-1}A^T + Q
$$

Update

Update Kalman gain, where $H$ is the observation model and $R$ the measurement noise covariance.

$$
K_k = P_{k|k-1}H^T (HP_{k|k-1}H^T + R)^{-1}
$$

Update state vector, where $z_k$ is the observed data.

$$
x_k = x_{k|k-1} + K_k(z_k - Hx_{k|k-1})
$$

Update covariance.

$$
P_k = (I - K_kH)P_{k|k-1}
$$
C Additional Figures

Figure 7: Option of frames after a round.

Figure 8: Result from choosing a frame.