1 Introduction

Object tracking is a pivotal technology in the realm of computer vision, gaining increasing importance due to its wide range of applications across various industries. This technology entails the automatic detection and monitoring of objects within a video sequence, interpreting their trajectories with high accuracy. The primary goal of object tracking is to maintain the identity of objects as they move through different frames of a video, ensuring that each object is accurately followed throughout the entire sequence. We present a hybrid object tracking model that utilizes YOLOv11 within a physics-based setup that uses some initial frames to predict the trajectory of the object, allowing us to track it with more accuracy and efficiency. We trained and tested the model using a dataset of free kick videos, which provided diverse and challenging scenarios for tracking moving objects. Results suggest high accuracy comparable to the current powerful models available coupled with faster and more energy-efficient performance.

1.1 Motivation

The motivation behind object tracking is deeply rooted in its vast applicability and potential to enhance numerous fields. In robotics, for instance, object tracking is essential for navigation, manipulation, and interaction with dynamic environments. Robots rely heavily on object tracking to understand their surroundings and make informed decisions. This capability extends to surveillance systems, where tracking individuals or objects in real-time can significantly improve security and monitoring. In the medical field, object tracking aids in analyzing patient movements and detecting anomalies. With the advent of autonomous vehicles, object tracking is essential for detecting and predicting the behavior of pedestrians, vehicles, and other dynamic elements in the environment.

1.2 What is Object Tracking?

Object tracking involves the use of algorithms to follow objects through consecutive frames of a video. The process begins with object detection, where the initial positions of objects are identified. These objects are then assigned unique identifiers, which the tracking system uses to monitor their movements across frames. The challenge lies in accurately maintaining these identifiers despite changes in object appearance, occlusions, and complex motion patterns.

1.3 Current Methods of Object Tracking

Various methods are employed for object tracking, each with its strengths and limitations. Traditional methods often rely on feature-based techniques, such as the Kanade-Lucas-Tomasi (KLT) tracker, which uses optical flow to track feature points. However, these methods can struggle with complex scenarios involving significant changes in object appearance or background clutter.

In contrast, modern approaches leverage deep learning techniques, offering enhanced accuracy and robustness. Convolutional Neural Networks (CNNs) have become the cornerstone of many state-of-the-art tracking algorithms. Models like the Fast R-CNN and

Figure 1: Object tracking

Faster R-CNN are particularly notable for their ability to balance detection speed and accuracy. Other advanced models, such as the DeepSORT algorithm, integrate appearance information to improve tracking performance, especially in the presence of occlusions.

Single Object Tracking (SOT) and Multiple Object Tracking (MOT) are two primary categories within the field. SOT focuses on tracking a single object, initializing the object in the first frame and following it through subsequent frames. MOT, on the other hand, involves tracking multiple objects simultaneously, requiring more complex data association techniques to maintain accurate trajectories for each object.

1.4 Challenges in Object Tracking

Several challenges complicate the task of object tracking.[1] These include:

1. **Training and Tracking Speed**: Balancing the accuracy and speed of tracking algorithms is critical, especially for real-time applications. Techniques such as the use of anchor boxes, feature maps, and feature pyramids help address these issues.

2. **Background Distractions**: Complex or cluttered backgrounds can make it difficult for tracking algorithms to distinguish between the object and the background, leading to potential errors.

3. **Multiple Spatial Scales**: Objects can vary in size and aspect ratio, complicating the tracking process. Techniques like image pyramids and feature pyramids are employed to handle these variations.

4. **Occlusion**: When objects overlap or are partially hidden, tracking algorithms may lose track of the object. Occlusion sensitivity measures help mitigate this problem by identifying critical image regions for classification.

1.5 Approach

Building upon the existing methodologies, my approach to object tracking introduces a physics-based model that integrates motion prediction and adaptive learning. This model

[1] Ref: https://viso.ai/deep-learning/object-tracking/

leverages physical principles to predict the future positions of objects more accurately. By combining these predictions with adaptive learning algorithms, the model can dynamically adjust to changes in object behavior and appearance, significantly improving tracking robustness and accuracy. Current benchmark object tracking models generally employ brute-force methods; take, for example, the YOLO model, which attempts to locate the object of interest within each frame and link its positions between successive frames. YOLO takes in 640×640 images, which presents a challenge when it comes to object tracking within a wide field of vision. With images of a larger size, it is forced to either shrink the image, which leads to a loss of information, or separate the image into individual 640×640 images, which is inefficient and time-consuming. Therefore, we suggest an approach that utilizes the YOLO model as a base and implements a trajectory prediction layer that provides estimations for the position, velocity, and acceleration of the object. This allows us to cut out an image of the appropriate size from each frame on which we can apply the YOLO algorithm, allowing for more accurate, efficient localization.

2 Theory

2.1 Pinhole camera model

The most common imaging model used in computer vision is the pinhole camera model[2], which is illustrated in Fig. 2.

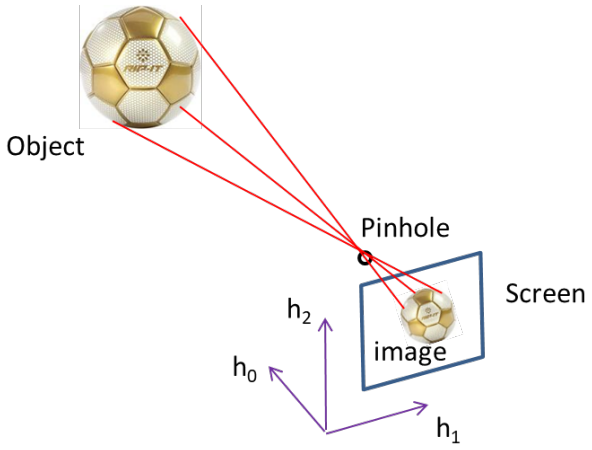

Figure 2: Pinhole camera model

Without loss of generality, suppose the pinhole is located at $(0, 0, 0)$. Denote the position of an object by $\vec{r} = (x, y, z)$, its velocity by $\vec{v} = (v_x, v_y, v_z)$, and its acceleration by $\vec{a} = (a_x, a_y, a_z)$. The screen on which the image is captured is located at a vector $\vec{h_s} = h_s\hat{h_0}$ from the origin. The screen coordinates basic vectors are $\hat{h_0}$, $\hat{h}_1$ and $\hat{h}_2$. That is, $< \hat{h_0}, \hat{h_1}, \hat{h_2} >$

[2]David A. Forsyth and Jean Ponce (2003). Computer Vision, A Modern Approach. Prentice Hall. ISBN 0-12-379777-2.

also form a right-hand coordinate system. Note that

$$\hat{h_0} \perp \hat{h_1} \perp \hat{h_2} \tag{1}$$

$$|\hat{h_0}| = |\hat{h_1}| = |\hat{h_2}| = 1 \tag{2}$$

The screen plane is given by

$$h_s\hat{h_0} + a\hat{h_1} + b\hat{h_2}, \quad a, b \in R \tag{3}$$

Then the projection of $\vec{r}$ on the screen, which is the image of the object, $\vec{p} = (a, b)$, is given by

$$-k\vec{r} = h_s\hat{h_0} + a\vec{h_1} + b\vec{h_2} \tag{4}$$

where k is the distance of the object from the pinhole and a and b are the "screen coordinates" of its image.

$$\left(\vec{r}, \hat{h_1}, \hat{h_2}\right) \begin{pmatrix} k \\ a \\ b \end{pmatrix} = -h_s\hat{h_0} \tag{5}$$

or

$$\left(\hat{h_1}, \hat{h_2}, \vec{r}\right) \begin{pmatrix} a \\ b \\ k \end{pmatrix} = -h_s\hat{h_0}$$

Define

$$C^{-1} = \left(\hat{h_1}, \hat{h_2}, \vec{r}\right)^{-1} \tag{6}$$

as the inverse camera matrix. The screen coordinates of the object (a, b), as well as its distance k from the pinhole, are given by

$$\underline{\begin{pmatrix} a \\ b \\ k \end{pmatrix} = -C^{-1}h_s\hat{h_0}} \tag{7}$$

2.1.1 The characteristics of the screen coordinate system

The world coordinates are denoted by $(\hat{x}, \hat{y}, \hat{z})$, where $\hat{z}$ is the upward direction and $(\hat{x}, \hat{y})$ spans the ground. In the real world situations that we hope to analyze, not only do the objects move, but the camera also moves simultaneously. Therefore, $< \hat{h_0}, \hat{h_1}, \hat{h_2} >$ translates and rotates around the pinhole, and the pinhole also moves. In typical scenarios, the camera stands vertically, so $\vec{h_2}$ is close to vertical, the angle between them being θ. Then

$$\hat{h_2} \cdot \hat{z} = \cos\theta \approx 1 \tag{8}$$

2.1.2 The solution to the imaging equation

In Eq. (4)

$$k\vec{r} + h_s\hat{h_0} + a\vec{h_1} + b\vec{h_2} = 0$$

Multiplying the equation by $\hat{h_i}$ for $i = 0, 1, 2$, noting that

$$\hat{h}_i \cdot \hat{h}_j = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}$$

We get

$$k\left(\vec{r} \cdot h\right) = -h_s$$

$$k\left(\vec{r} \cdot \hat{h_1}\right) + a = 0$$

$$k\left(\vec{r} \cdot h_2\right) + b = 0$$

Therefore

$$\underline{k = -\frac{h_s}{\vec{r} \cdot h}}, \quad a = -k\left(\vec{r} \cdot h\right) = \underline{h_s \frac{\vec{r} \cdot \hat{h_1}}{\vec{r} \cdot h}}, \quad b = \underline{h_s \frac{\vec{r} \cdot \hat{h_2}}{\vec{r} \cdot \hat{h_0}}} \tag{9}$$

The image is at (a, b) on the capturing device, typically a CCD or CMOS image sensor. Note that the final image shown in an video will be translated and zoomed according to the frame configuration. The final location of the pixel is

$$\left(a', b'\right) = (c_x, c_y) + M\left(a, b\right) \tag{10}$$

where (c_x, c_y) are the offset of the image sensor center to the video center and M is the zoom factor.

Note that if $\vec{r} \perp \hat{h_0}$,this object is out of the FOV (field-of-view) and not on the screen.

Similarly, if $\vec{r} \cdot \hat{h}_0 < 0$, the object is "behind" the camera and cannot form an image.

2.2 The bounding box of an object

Assuming the object's diameter is D, the whole shape can be approximated by a box whose vertices are given by

$$\vec{r_i} = \vec{r} + \left(\pm\frac{D}{2}, \pm\frac{D}{2}, \pm\frac{D}{2}\right) \tag{11}$$

where $1 \leq i \leq 8$. Then for each $\vec{r}_i$, its projection on the camera screen is $\vec{p}_i = (a'_i, b'_i)$, also obtained by Eq. (9).

Thus, the bounding box of the object's image is the range of the 8 projections:

$$ll_x = \min(a'_i), \quad ll_y = \min(b'_i), \quad 1 \leq i \leq 8 \tag{12}$$

$$ur_x = \max(a'_i), \quad ur_y = \max(b'_i), \quad 1 \leq i \leq 8 \tag{13}$$

where ll, ur denote the lower-left and upper-right corner, respectively. For the sake of simplicity, denote the bounding box of the object's image by

$$\vec{B} = \begin{pmatrix} ll_x \\ ll_y \\ ur_x \\ ur_y \end{pmatrix} = \begin{pmatrix} \min a'_i \\ \min b'_i \\ \max a'_i \\ \max b'_i \end{pmatrix} \tag{14}$$

2.3 Motion model

Denote the state of an object at the time t by

$$\vec{s}(t) = \begin{pmatrix} \vec{r} \\ \vec{v} \\ \vec{a} \end{pmatrix} \tag{15}$$

Then after a short time Δt, its state becomes

$$\vec{s}(t+\Delta t) = \begin{pmatrix} \vec{r} + \vec{v}\Delta t + \frac{1}{2}\vec{a}\,(\Delta t)^2 \\ \vec{v} + \vec{a}\Delta t \\ \vec{a} + \Delta\vec{a} \end{pmatrix} \tag{16}$$

The bounding box of its image changes to

$$\vec{B}\,(t) = f\,(\vec{r}) \rightarrow \vec{B}\,(t+\Delta t) = f\left(\vec{r} + \vec{v}\Delta t + \frac{1}{2}\vec{a}\,(\Delta t)^2\right) \tag{17}$$

where $f\,(.)$ is the operator that maps the 3D location of an object to the bounding box of its image, described by Eq. (9) - (14).

3 Trajectory prediction

Given a sequence of image frames captured by a video camera, we must predict the trajectory of an object. As human or animals do, we first need to locate the object in these frames and then predict its next location based on its current trajectory. Very often, we will need to use experience on how such objects typically move.

YOLO (You Only Look Once) is a real-time object detection and image segmentation deep neural network (DNN) model, developed by Joseph Redmon and Ali Farhadi at the University of Washington[3]. Since its launch in 2015, YOLO has gained popularity for its high speed and accuracy. The latest model version is YOLOv11.[4] This pre-trained model can detect 80 classes, on of which is "sports ball" (class 32). In this project, I use YOLOv11s to locate the bounding box of a soccer ball, shown in Fig 3.

[3]https://docs.ultralytics.com/

[4]https://huggingface.co/Ultralytics/YOLO11

Figure 3: The red box is the bounding box of the soccer ball

3.1 Assumptions

As a first-order approximation, assume that the video camera does not move, and that it is perfectly aligned with the world coordinate system. That is

$$\left(\hat{h}_0, \hat{h}_1, \hat{h}_2\right) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

where the unit is meters. The distance of the camera screen to the optimal center (pinhole) is $h_s = 0.2$ meters. The camera offset and zoom in Eq. (10) are taken as

$$(c_x, c_y) = (0, 0) \quad M = 1000$$

The camera is assumed to be 1.5 m above the ground, which means the ground is at $z = -1.5$.

These numbers are rather arbitrary, but as shown later, they have little effect on prediction accuracy.

The diameter of a size-5 soccer ball is taken as 0.22 m in this project, which is the international standard.[5]

The time step between two consecutive video frames is $\Delta t = 0.03$ sec, corresponding about 30 frames per sec.

3.2 Algorithm

Given $n + 1$ bounding boxes in the corresponding consecutive images, how do we predict the bounding boxes in the next m image frames? Our observations are the bounding boxes,

[5] https://theifab.com/laws/latest/the-ball/#qualities-and-measurements

and the unknowns ("hidden states") are the states, $\vec{s} = (\vec{r}, \vec{v}, \vec{a})$, of the object. If we can estimate the states, we can extrapolate the motion to the next several time steps.

As an approximation, assume acceleration in this sequence is a constant. So all the states at $t = 0, \Delta t, 2\Delta t, ..., n\Delta t$, are determined by the initial state. We only need to estimate the initial state. The imaging process is illustrated in Fig. 4.

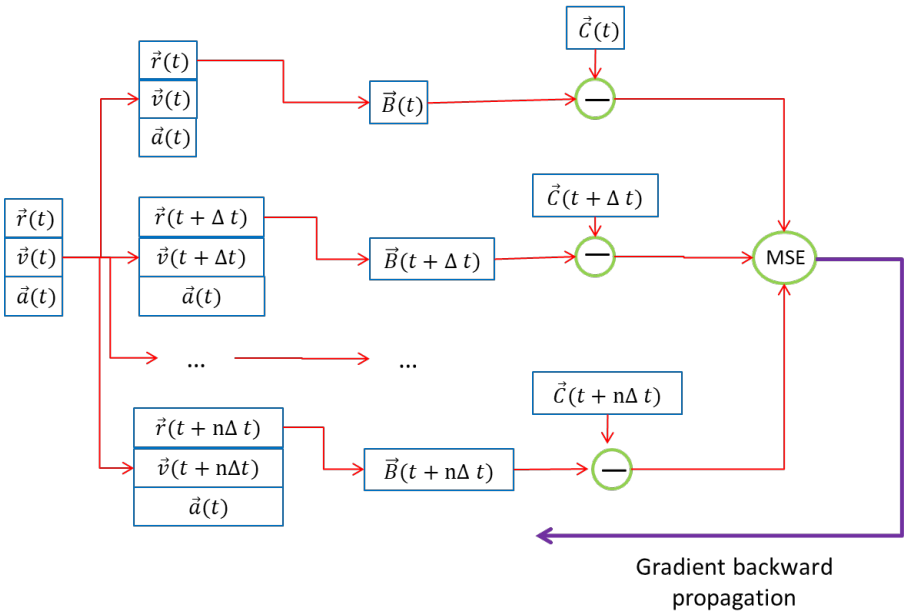

Figure 4: The flow of imaging process, which is equivalent to a neural network. Then the estimation of the original state becomes optimizing the loss function through gradient descent method.

The steps of the above imaging process are given by Eq. (9) - (14). Note that these equations are all differentiable, thus the process is equivalent to a 2-layer neural network. The 1st layer maps $\vec{s}(t)$ to $\vec{s}(t + i\Delta t)$, $i = 0, 1, .., n$. The 2nd layer maps each $\vec{s}(t + i\Delta t)$ to corresponding boxes $\vec{B}(t + i\Delta t)$. Then compare all $\vec{B}(t + i\Delta t)$ with the actual observations $\vec{C}(t + i\Delta t)$ to get the MSE (mean squared error) loss:

$$L = \sum_{i=0}^{n} ||\vec{B}(t + i\Delta t) - \vec{C}(t + i\Delta t)||^2 \tag{18}$$

Then the problem is to find initial $\vec{s} = (\vec{r}, \vec{v}, \vec{a})$ to minimize L. Gradient descent (GD) is one of the most popular methods to solve such optimization problems.[6] The key concept is based on the gradient vector of partial derivatives. The loss function has a gradient vector:

$$\frac{\partial L}{\partial \vec{B}_i} = \left(\vec{B}(t + i\Delta t) - \vec{C}(t + i\Delta t)\right), \quad i = 0, 1..., n \tag{19}$$

[6]Boyd, Stephen; Vandenberghe, Lieven (2004-03-08). Convex Optimization. Cambridge University Press. doi:10.1017/cbo9780511804441. ISBN 978-0-521-83378-3.

where $\vec{B}_i = \vec{B}(t + i\Delta t)$. It gives the steepest direction along which the value of L changes. So we can change $\vec{B}_i$ by a small amount

$$\delta\vec{B}_i = -\lambda\left(\frac{\partial L}{\partial \vec{B}_i}\right)$$

where λ is called the learning rate. Then L is reduced. Similarly, $\delta\vec{B}_i$ is distributed to those variables that determine $\vec{B}_i$,namely, $\vec{r}_i = \vec{r}(t + \Delta t)$. Eventually the initial state $\vec{r}, \vec{v}, \vec{a}$ all receives their corrections. Then we use the new $\vec{s}$ to do the imaging again ("forward pass"), and correct all variables again ("backward propagation"). Ideally, these iterations stop when

$$\frac{\partial L}{\partial \vec{B}_i} = 0, \quad i = 0, 1...n \tag{20}$$

which indicates a local minimal. In practice, we stop after a certain of iterations or the loss is below a threshold.

Thanks to the rapid advances of deep learning (DL), there are plenty of mature and powerful frameworks to do GD. For this project, I chose PyTorch, the most popular open source DL platform developed by Meta.[7] Their website provides a nice tutorial on using PyTorch as an optimization tool:

https://pytorch.org/tutorials/beginner/basics/optimization_tutorial.html

Once the initial state has been estimated, the imaging process in Fig. 4 is repeated for time step $n + 1$,..., $n + m$ to predict the next m bounding boxes.

3.3 Test of simulated motion

To validate the prediction method, I generated simulated motions. This is the simulation process:

1. A ball has a location (x, y, z), and initial velocity (v_x, v_y, v_z), which are randomly generated. But $v_z \geq 0$.

2. At each time step, this ball experiences 3 accelerations. One is friction, slowing its speed by a faction. Another is perpendicular to its velocity, changing its direction in horizontal plane. The third is gravitational acceleration. The first are randomly generated, but can not be large.

3. If the ball hits the ground ($z = -1.5$), it will bounce.

4. At each step, calculate its bounding box.

5. Applying $n+1$ consecutive boxes to predict the next m boxes, and calculate the error from the actual simulated m boxes.

The random accelerations are still applied to the next m images, so the error between prediction and simulation is expected. In this simulation, I used $m = 4$.

Fig. 5 compares the prediction with the simulated ball movement. In 3D, they are quite different. But in "video" (2D), they are very close. This is not surprising, because pinhole

[7] https://pytorch.org/

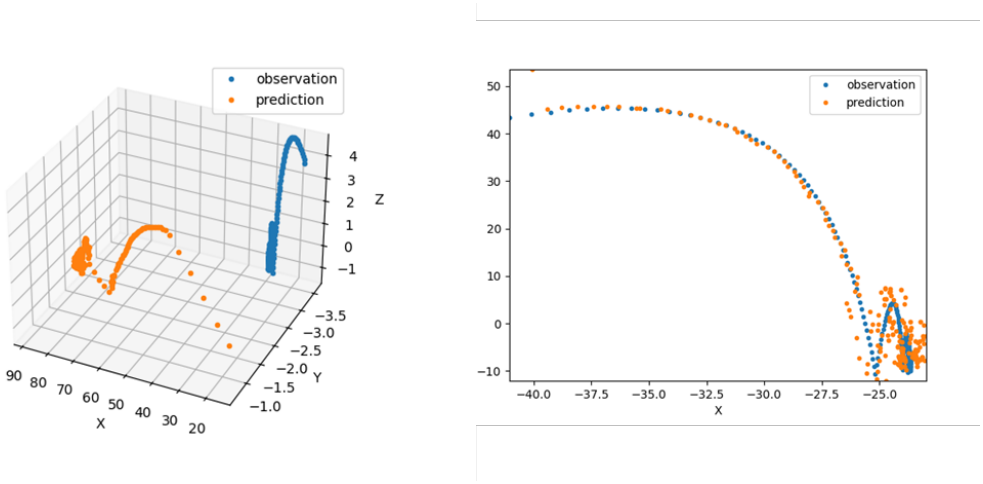

Figure 5: The trajectory of the predicted and simulated (observed) motion in physical 3D world (left) and in video (right)

camera model is a single-eye vision system, lacking depth perception. There are infinite number of 3D points that project to the same imaging point.

The squared root of MSE at every time step is plotted in Fig. 6. Initially there is a huge error,, but it drops very rapidly. Then the error is close to 0 pixel. After that, the ball bounces off the ground multiple times. Each time the error jumps and then drops. This is also not surprising, because bouncing is not considered in the prediction. Even with bouncing, the error is no more than 3 pixels, proving the high accuracy of this model.

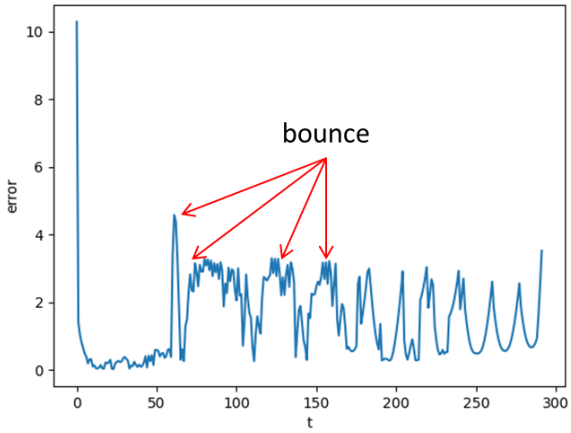

Figure 6: The error $\sqrt{\left(a'_{predict} - a'_{sim}\right)^2 + \left(b'_{predict} - b'_{sim}\right)^2}$ vs. time step

3.4 Application to the real scenes

I applied this predictor to a video with many free kicks. And the results are very good. Fig. compares the prediction and observation in some frames. The complete video is at https://github.com/zcheng10/pdl/blob/main/test/ext_clip_0_boxed.mp4.

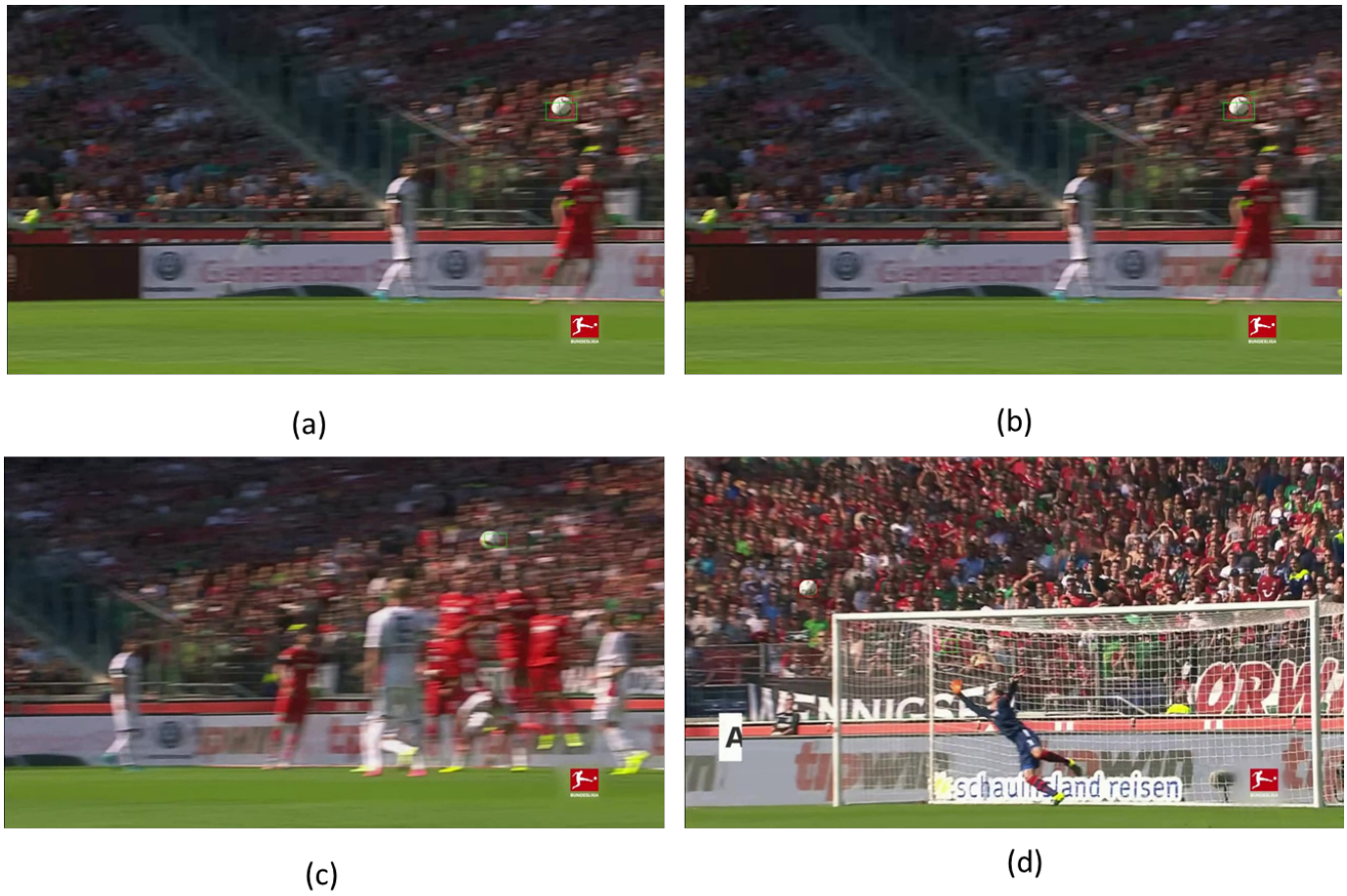

Figure 7: Observation (red boxes) and prediction (green boxes) in some video frames

Note that YOLO11 only takes image size 640×640, so the input images have to be compressed. Thus in many of fast-moving frames, the ball could not be detected, which hinders the prediction. Nevertheless, the predictions were accurate even when there are no detected bounding boxes in several consecutive frames, shown in Fig. 7 (c).

Fig. 7 (d) shows another interesting finding. Due to lack of detected boxes in the previous frames, the prediction (green box) is ahead of the observation (red box). But it accurately predicted where the ball would fall. In fact, after several frames, the ball fell to the almost exact location.

Since we can obtain fairly accurate prediction, we can focus the detection of objects in the predicted location, rather than searching the whole image. The detection speed and accuracy can therefore be greatly improved.