Uplifting Table Tennis: A Robust, Real-World Application for 3D Trajectory and Spin Estimation

Supplementary Material

A. Training Details

The front-end modules are trained with a learning rate of 10^{-3} using the Adam optimizer [23] and a batch size of 4. For ball detection, we apply random flipping, rotation, translation, cropping, and color jittering as augmentations. For table keypoint detection, we do not use flipping, as it would change the semantics of the keypoints. Instead, we apply an additional random perspective transformation to simulate different camera angles. Especially for table keypoint detection, we found that augmentations are crucial to achieve generalization ability since the TTHQ dataset is diverse but the number of annotated frames is still limited. We save the model with the best $ACC@5$px metric on the validation set during training and use it for testing. Similar to 2D human pose estimation, we simply train the models to predict gaussian heatmaps centered at the ground truth ball and keypoint positions. The ground truth heatmaps are created at a resolution of 1920×1080, and we use $\sigma = 6$px for the Gaussian. Comparing the predicted and ground truth heatmaps, we use a simple L_2 loss. All evaluations are performed at a resolution of 1920×1080 and the $ACC@X$px metric is also computed at this resolution.

The ball detection backbones are first initialized with ImageNet [7] (Segformer++ and WASB) or MAE [17] (VitPose) weights, then pre-trained on the Blurball dataset [14], and finally fine-tuned on the TTHQ training set. The table keypoint detection backbones are also initialized similarly, but only fine-tuned on the TTHQ training set, as the Blurball dataset does not contain sufficiently diverse table keypoint annotations.

For the training of the back-end module, we stick closely to the training procedure of *Kienzle et al.* [22], but we incorporate additional augmentations of the synthetic training data. We simulate missing detections by randomly dropping 5% of the ball and table keypoint detections during training. Moreover, we randomly sample the frame rate between 20 FPS and 60 FPS during training. A further difference to the *Kienzle et al.* training procedure is that we use our extended synthetic dataset, which contains 140000 trajectories instead of 50000 and includes more diverse match scenes like serves. In contrast to *Kienzle et al.*, we do not validate on the synthetic data, but instead validate on the validation set of the TTST dataset [22]. We choose the final model as compromise between the best F_1 score for spin prediction and the best m2DRE for trajectory prediction. If multiple epochs achieve the same maximum F_1 score, which happens regularly, we choose the one with the best m2DRE among them. To allow for a meaningful interpretation of the m2DRE metric, we evaluate all results at a resolution of 1920×1080, which ensures compatibility with our front-end evaluations. Our models predict the spin in the global world coordinate system, but for evaluation and extraction of the spin class, we transform the spin into the local ball coordinate system as described in [22].

Figure 7. Ball detection filtering. The green dots are the valid detections of the Segformer++ (B2) model after filtering. The crimson dots are the false positives of the Segformer++ (B2) model that were removed. The orange dots are the predictions of the WASB model for the filtered frames.

B. Architectures

Front-End Architectures: To ensure a fair comparison of front-end architectures, we tuned 2 significant parameters: The input resolution and the model size. We chose these

parameters such that each model can be trained on a single Nvidia V100 GPU with 32 GB of memory. Moreover, the models are tuned such that their inference speed is as similar as possible. We provide the exact parameters in Table 1 and Table 2.

Back-End Architecture: We stick closely to [22] when designing the back-end architecture. We use an embedding size of $d = 128$ and $L = 16$ transformer layers with 4 attention heads each. The Spin Head and Location Head both consist of 3 fully connected layers with ReLU activations in between.

Positional Encoding: Unlike standard transformer-based architectures that rely on a fixed positional encoding based on the token's index, our system must handle inputs with varying time stamps and missing detections. To address this, we use a custom Rotary Positional Embedding (RoPE) that encodes the exact time information of each detection into the attention mechanism. The core of our implementation is a modification to the standard RoPE approach, where we replace the discrete position index n with a highly granular, discretized timestamp p_n:

$$p_n = \text{round}(t_n / \Delta t) \tag{2}$$

with $\Delta t = 2\,\text{ms}$. This discretization is so fine-grained that it effectively represents the exact time for all realistic video frame rates. By replacing n with p_n in the original RoPE formulation [36], we obtain the following rotation matrix

$$R = \begin{pmatrix} \cos(p_n \cdot \theta_m) & -\sin(p_n \cdot \theta_m) \\ \sin(p_n \cdot \theta_m) & \cos(p_n \cdot \theta_m) \end{pmatrix} \tag{3}$$

where $\theta_m = \frac{1}{10000^{2m/d}}$ is the angle, d is the feature dimension and m indexes the feature dimensions. For a pair of elements from the feature vector $\mathbf{x} = (x_1, ..., x_d)$, the rotated components are computed as:

$$\begin{pmatrix} x'_{2m} \\ x'_{2m+1} \end{pmatrix} = \begin{pmatrix} \cos(p_n \cdot \theta_m) & -\sin(p_n \cdot \theta_m) \\ \sin(p_n \cdot \theta_m) & \cos(p_n \cdot \theta_m) \end{pmatrix} \begin{pmatrix} x_{2m} \\ x_{2m+1} \end{pmatrix}. \tag{4}$$

This time-based encoding allows the attention mechanism to inherently learn the temporal relationships between detections, regardless of their spacing in the input sequence.

C. Ball & Table Keypoint Filtering

The back-end stage of our pipeline is designed to be robust against missing ball and table keypoint detections. However, it is relatively sensitive to false positive detections, which is why we apply a filtering step to the raw detections of the front-end. For ball detections, we compare the predictions of the Segformer++ (B2) model with the predictions of the WASB model and only keep detections with a maximum distance of 20 px to a WASB detection. Because the model architectures are fundamentally different, they usually do not make the same false positive detections, allowing us to filter them out with very high precision. We illustrate this filtering step in Figure 7 for three different example trajectories in the TTHQ test set.

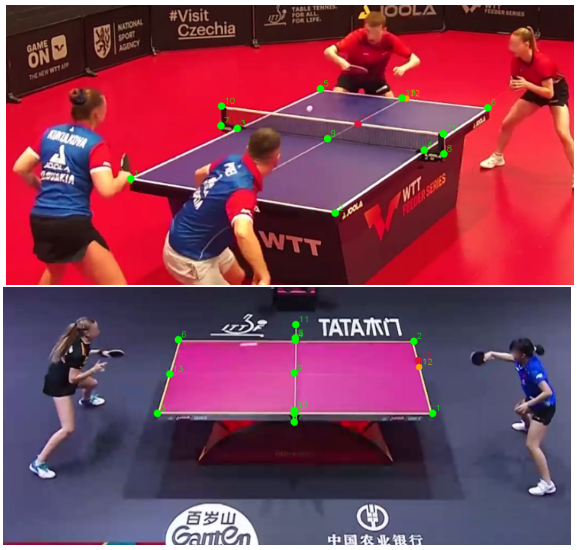

Figure 8. Table keypoint detection filtering. The green dots are the valid detections after filtering. The crimson dots are the false positives of the Segformer++ (B2) model that were removed. The orange dots are the predictions of the WASB model for the filtered frames.

We use this two-model based filtering approach instead of more established filtering methods like Kalman filtering or polynomial fitting, because they fail in a significant failure case: Sometimes, the feet or paddle movements of the players are falsely detected as balls for multiple consecutive frames. These false positive detections cannot be easily differentiated from real ball trajectories, which is why the established filtering methods fail. However, our two-model based filtering approach is very robust against this failure case, resulting in a very high precision of the final ball detections.

For table keypoint detections, we apply a similar filtering step. We only keep the detections of the Segformer++ (B2) model that are within a distance of 10 px to a WASB detection. Additionally, we utilize the static nature of the table keypoints and apply the DBSCAN clustering algorithm [9] to the remaining detections of each keypoint over the entire trajectory. The final filtered detections are then the cluster centers of the largest cluster for each keypoint. This results in an extremely robust filtering of false positive detections. We illustrate this filtering step in Figure 8 for two different example trajectories in the TTHQ test set.

D. Comparison of Datasets

We provide a comparison of existing table tennis datasets in Table 5.

Dataset	Resolution	# Balls	# Table Keypoints	# Spin Class
TTHQ (ours)	1920×1080	9092	257	57
TTST [22]	1920×1080	1197	50	50
Blurball [14]	1280×720	50000	50000*	0

Table 5. Comparison of table tennis datasets. We report the video resolution, number of frames with ball annotations, number of frames with table keypoint annotations, and number of trajectories with spin class annotations. The * indicates, that table keypoints are not annotated, but can be generated from the provided homography. However, while many frames with table annotations can be obtained this way, the variety of the frames is extremely low, because the dataset is sourced from a limited number of videos.

The **Blurball dataset** [14] has the largest number of ball annotations, and using the provided homographies, pseudo table keypoints can be generated. As the dataset is sourced from 24 different videos, the table keypoints lack diversity and are, thus, not suitable for training a robust table keypoint detection model. However, since ball detection training data needs to capture a wide variety of conditions due to the dynamic nature of the ball, the Blurball dataset is still a valuable resource for training ball detection models. Due to the low resolution of the videos, we do not use this dataset for fine-tuning, but utilize it for pre-training our ball detection models.

The **TTST dataset** [22] has a high resolution and provides annotations for balls, table keypoints, and spin. Because the scenes are not very diverse and the number of annotated frames is relatively low, the dataset is not suitable for training robust front-end models. However, it is still a valuable resource for validating the performance of our pipeline. Especially valuable are the dense annotations for each trajectory, which allow us to evaluate the performance of the back-end model independently of the front-end.

Our **TTHQ dataset** provides high resolution videos with a high variety of scenes and numerous annotations for balls, table keypoints, and spin. This makes it the first dataset that is suitable for training robust front-end models. Moreover, we can also utilize the dataset for the validation of the spin predictions. While the annotations are not dense like in TTST, the high variety of scenes and situations makes it an interesting benchmark for evaluating the overall performance of the full pipeline.

E. Reproducibility and Open Resources

To foster reproducibility and facilitate further research in the field, we provide our code at
https://kiedani.github.io/WACV2026/.