Estimating 3D ball trajectories from single camera padel videos

Benjamin de Charmoy
Labs
Thinkst Applied Research
Cape Town, South Africa
benjamin@thinkst.com

Abstract— Extracting useful information from videos of padel games The estimnation of objects world coordinates using a single camera is a inherently ill posed problem. The measurement space 2D in pixel units and the state space is 3D in meters. This paper explores the use of prior knwoledge about the scene and object dynamics to produce estimates. The padel court scene has many known geometric features such as court markings, net hight, wall (glass) and fence hights and these are exploited to perform both camera calibration and pose estimation. Robust mappings from 3D world coordinates to 2D image plane coordinates

Index terms—Padel analytics, State estimation, Single image camera calibration, Tracking

I. Introduction

Padel is a racquet sport played by teams of two in a 10m by 20m court.The court has glass walls and the ball can be played off and onto the wall. Scoring works similarly to tennis and matches last between one and two hours. During this time the players and a ball move around in the court see Figure 1. This movement is captured by a single broadcast camera at approximatly thirty frames per second. This work setout to unwind those events. Given these video recordings can we reconstruct the events that took place with the goal to extract useful player metrics.

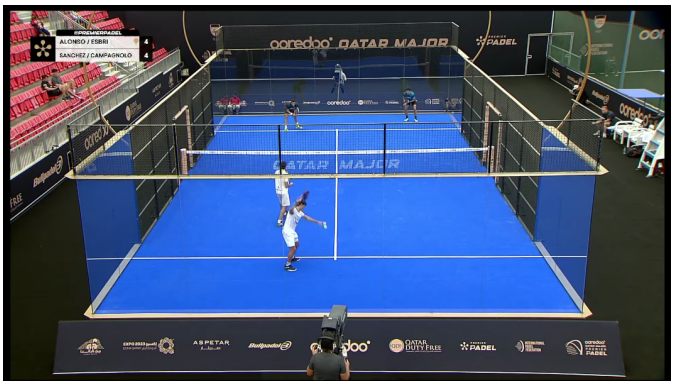

Figure 1: A typical broadcast view of game play.

To achieve this 4 tasks were identified: single image camera calibration and camera pose estimation; image plane object detection; world coordinate frame object tracking and data visualizations.

As part of this project a dataset CourtVisionPadelDataset was compiled from videos from the *Ooredoo Qatar Major 2023* tornament. This dataset consistes of TODO:X hours of live broadcast. It's annotated to give timestamps of rallies and bounding boxes of the ball. Accompanying the dataset is Python library CourtVision that implements the techniques desribed in this paper. CourtVision uses *Kornia* [1] a *Pytorch* [2] based computer vision library.

A. Paper overview

TODO:

II. Unwinding events

A. Single image camera calibration

Camera calibration is the process of estrimating both the intrinsic and extrinsic camera parameters. The intrinsic parameters K (camera matrix) and d (distortion cofficients) together with the pinhole camera model determine how rays are projected onto the sensor. The extrinsics R (rotation matrix) and t (translation vector) relates points in the world coordinate system to the camera coordinate system. Without access to the camera, calibration was performed by exploiting known scene geometry. Using the Padel court specifications Figure 2 twenty six point correspondences were identified.

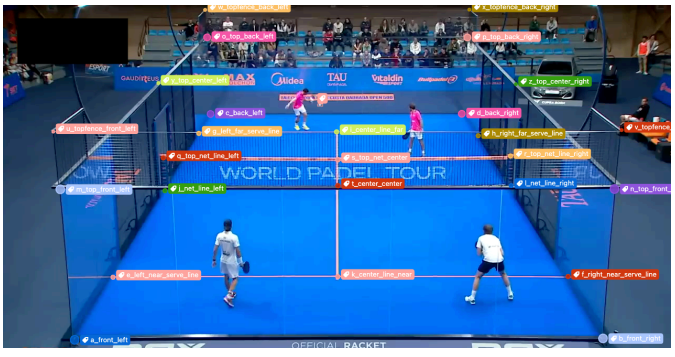

Figure 2: Twenty six points used for camera calibration and camera pose estimation.

Zhang's calibration approach assumes camera coordinates all lie on the same plane. The padel court scene has many co-planar points. For example all points on the floor or all points on the front glass. In total eight such planes were used.

The manual annotation process included an unquantified amount of noise. To eleaviate CourtVisions' implementation added gaussian noise to each and did an exhausative search for the best K and d. The mean reprojection error was used to determine the best camera matrix and distortion coeeficeints. Using this method a reprojection error of 3.50 pixels in images of resolution $1920x1080$.

A similar set of point correspondences were used to estimate the pose of the camera relative the court. Estimating camera pose from known point correspondeces was done in the traditional method using OpenCV's solvepnp implementation. This resulted in a reprojection error of 9.78 pixels. TODO: quantify this in meters!!

B. Image plane object detection

The task of detecting players and the ball in each frame is a well understood problem. There are a plethora of off-the-shelf models that can be finetuned to perform this task. Howerver, the datasets available in the domain of padel is limited. To this end a cyclic pipeline of train, evealuate, label, train was established. For detecting the players a Yolov8 [3] model was used. Ball detection uses a Faster-RCNN [4] model. All detections were passed to the tracker (see Section II.C) which did player re-identification and tracking. The dataset used for fintuning these models is made available [5].

C. World coordinate frame object tracking

Image plane object detection provides pixel locations of the ball and players per frame. Camera calibrartion gave a mapping from world coordinates to image plane pixel locations. By combining these two results and exploiting the dynamics of a ball in flight estimating the world coordinates of the ball is possible. The player tracking is done using the assumption that they remain on the court floor. This assumption is valid for the majority of the time. The ball tracking is done using a particle filter.

Particle filters [6] represent possible states of the ball as particles each with an associated weight. The full state is defined by a set of N particles where each particle n is a vector $\vec{X}_{\text{p=n}} = \{x, y, z, \dot{x}, \dot{y}, \dot{z}, \ddot{x}, \ddot{y}, \ddot{z}, w\}$. x is the position $\dot{x}$ is the velocity and $\ddot{x}$ is the acceleration in the $\hat{x}$ direction. Similarly for y and z. w is the probability mass at that location in the state space. The particles are initialized at random with x, y, x drawn from a uniform distribution over the cube making up the court. Ball speeds are approximately $20\,\text{km/h}$ to $60\,\text{km/h}$ during a serve in padel. Thus, velocity is drawn from a normal distribution with mean 0 and standard deviation $8.3\ [m\ /\ s]$. The acceleration is drawn from a normal distribution with mean 0 and standard deviation 0.1 with $\ddot{z} = -9.8$. The weight is set to $\frac{1}{N}$. The state is updated every $dt = \frac{1}{f}$ seconds where f is the frame rate of the video.

Particle filters are a sequential Monte Carlo method and have two steps *predict* and *update*. During the *predict* phase a new set of particles are sampled from the previous state according to:

$$p(x_{\text{t+1}} |\ z_{1:t}) = \int p(x_t|x_{\text{t-1}})p(x_{\text{t-1}}|z_{1:\text{t-1}})dx_{\text{t-1}} \tag{1}$$

The *predict* phase incorporates dynamics of the ball $p(x_t|x_{\text{t-1}})$ which propergates each particle according to the following pseudo code:

```
def predict(state: Tensor, dt: float) -> Tensor:

  noise = torch.randn_like(state) * motion_noise
  # Update position
  state[:, :3] += state[:, 3:6] * dt + noise
  # Update velocity
  state[:, 3:6] += state[:, 6:9] * dt + noise
  # Update acceleration
  state[:, 6:9]
```

During the *update* phase the particles are weighted according to the likelihood of the measurement z_t at time t. $p(z_t|x_t)$ is the likelihood of the measurement given the state. The likelihood is calculated weights updated using the following pseudo code:

```
 def update(state:Tensor, measurement: Tensor) ->
Tensor:
  # predicted measurement given the state
  pred_obs = state[:,:3]@world_to_cam@cam_to_image
 # Error predicted observation and the measurement
 error = torch.norm(pred_obs - measurement, dim=1)
  # The likelihood of the error
  likelihood = torch.exp(
    -0.5 * error ** 2 / measurement_noise
  )
  # Update weight using likelihood
  state[:, -1] *= likelihood
  # Normalize the weights
  state[:, -1] /= state[:, -1].sum()
  # Resample the particles
  state = resample(state)
```

The ball tracker outputs a state estimate for each frame. This is the mean of the particles weighted by their probability mass. The mean state is then projected back onto the image plane. Since the tracker holds N particles the uncertainty of the state can be estimated. The uncertainty is calculated as the standard deviation of the particles weighted by their probability mass.

D. Data visualizations

References

[1] E. Riba, D. Mishkin, D. Ponsa, E. Rublee, and G. Bradski, "Kornia: an open source differentiable computer vision library for pytorch," 2019. [Online]. Available: http://arxiv.org/abs/1910.02190

[2] A. Paszke, S. Gross, et al., "Automatic differentiation in pytorch."

[3] G. Jocher, A. Chaurasia, and J. Qiu, "YOLO by Ultralytics," 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[4] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: towards real-time object detection with region proposal networks," 2015. [Online]. Available: http://arxiv.org/abs/1506.01497

[5] b. de Charmoy, "CourtVision," 2023.

[6] A. Doucet, and A. Johansen, "A tutorial on particle filtering and smoothing: fifteen years later," *Handbook Nonlinear Filtering*, no. December, pp. 656–704, 2011, doi: 10.1.1.157.772. [Online]. Available: http://automatica.dei.unipd.it/tl_files/utenti/lucaschenato/Classes/PSC10_11/Tutorial_PF_doucet_johansen.pdf