Project KITTY:

Pose-Based Badminton Analytics and Feedback System

Taran Mohta

1 Problem Statement

While India continues to produce world-class athletes in sports like badminton, wrestling, and boxing, the nation lags behind global counterparts in leveraging sports science and AI-driven feedback systems. The performance gap at international competitions—such as the Olympics—isn't just due to physical preparation, but also a lack of access to cutting-edge, technology-integrated training tools. Countries leading the medal tables are rapidly adopting machine learning pipelines for pose correction, motion analysis, and strategic feedback—empowering their athletes with real-time, intelligent insights.

This project, undertaken under the banner of **Project KITTY** (Kinematic Intelligence and Tactical Training for Youth), aims to bridge that gap using computer vision and machine learning. Specifically, it focuses on badminton—a fast-paced, biomechanically rich sport—to demonstrate the feasibility of automated pose-based shot classification and corrective analytics. Through a structured pipeline involving keypoint extraction, biomechanical feature engineering, and threshold-based performance diagnostics, this work lays the groundwork for scalable, intelligent sports training tools tailored for Indian athletes.

2 Why Badminton?

Badminton holds strategic importance for India's Olympic ambitions, consistently contributing to the medal tally in recent Games. As a high-intensity, technically demanding sport, it sits at the intersection of speed, precision, and biomechanics—making it an ideal candidate for AI-powered motion analysis. Unlike sports such as football or basketball, where extensive research already exists, badminton remains an underexplored yet high-potential domain within sports tech. This project taps into that emerging intersection of biomechanics, computer vision, and machine learning—aiming to push the frontier of research and real-world application in badminton performance analytics.

3 Creating Data Set, Training Shot-Classifier, Creating Rule Based Feedback System

The analysis pipeline consists of three main steps: dataset creation with keypoint extraction and feature engineering, multiclass shot classification, and rule-based feedback generation. I describe each step below.

3.1 Step 1: Dataset Creation, Keypoint Extraction & Feature Engineering

To enable a robust and interpretable analysis pipeline, I began with the ShuttleSet dataset—an annotated badminton video collection curated under the CoachAI project [1]. Each video contained shot-wise metadata such as the player making the shot, shot type, and the impact frame. The goal of this step was to extract player-centric temporal windows and transform raw pose data into meaningful biomechanical features.

3.1.1 Frame Extraction

For every shot, 61 frames were extracted—spanning from 30 frames before to 30 frames after shuttle impact—using OpenCV. These frames capture the full movement arc of the player surrounding each shot.

3.1.2 Court-Side Cropping

To reduce visual clutter and focus on the relevant player, all frames were cropped to the near court side using fixed bounding-box coordinates determined by camera angle and consistent court geometry.

3.1.3 Pose Estimation using MediaPipe

Each extracted frame was processed with MediaPipe Pose, which returns 33 3D keypoints for the entire body. This provided frame-wise pose data for the player executing the shot.

3.1.4 Feature Engineering from Keypoints

From the 33 keypoints, I computed a set of handcrafted biomechanical features to capture execution quality:

- Elbow Angle: relative positions of shoulder, elbow, and wrist
- Wrist Velocity: temporal derivative of the wrist keypoint's coordinates
- Torso Lean: deviation of the torso vector from the vertical axis
- Knee Flexion Angle: assessment of stance and stability
- Arm Extension Ratio: measurement of shot reach
- Centroid Displacement: quantification of body movement and follow-through

I also computed keypoint statistics such as `kp_0_x_mean`, `kp_0_y_max`, and `kp_24_y_std` to capture temporal and spatial significance.

3.2 Step 2: Multiclass Shot Classification using XGBoost

With biomechanical features in place, the next step was to classify shot types using a supervised learning model.

3.2.1 Model Selection

I chose XGBoost for its performance on structured data, support for multi-class classification, built-in regularization, and efficient training [2].

3.2.2 Training Setup

I split the feature vectors into training and validation sets. The model was trained to classify shot types:

- Smash
- Drop
- Net shot
- Clear

I evaluated performance using accuracy, precision, recall, and F1-score.

3.2.3 Classification Report

	precision	recall	f1-score	support
clear	0.74	0.95	0.83	21
drop	0.50	0.25	0.33	8
net shot	1.00	1.00	1.00	17
smash	0.90	0.75	0.82	12
accuracy			0.83	58
macro avg	0.79	0.74	0.75	58
weighted avg	0.82	0.83	0.81	58

3.3 Step 3: Rule-Based Feedback using Interquartile Thresholding

Beyond shot classification, I implemented a rule-based engine to generate automated feedback on biomechanical performance.

3.3.1 Rationale

While the classifier predicts "what" the shot is, the rule-based engine interprets "how well" it was performed.

3.3.2 Interquartile Range (IQR) Method

I computed Q1, Q3, and IQR for each feature, then flagged values outside [Q1–1.5×IQR, Q3+1.5×IQR]. For example, an elbow angle below the lower bound during a smash may indicate incomplete arm extension.

Table 1: Example of Shot Type, Feedback Inference, Features, Thresholds, and Rationale

Shot Type	**Feedback Inference**	**Features Involved**	**Threshold**	**Rationale**
Clear (0)	Push up through your knees and hips—stand taller as you swing so the shuttle clears deeper.	`kp_26_y_max`, `kp_23_y_max`, `kp_24_y_max`	kp_26_y_max: [0.556, 0.736]; kp_23_y_max: [0.482, 0.636]; kp_24_y_max: [0.475, 0.651]	Knee (kp_26) and hips (kp_23, kp_24) vertical positions reflect leg extension crucial for vertical push and generating lift.
Smash (1)	Drive your elbow higher and extend fully—finish the shot with a strong follow-through.	`elbow_angle`, `kp_31_y_mean`	elbow_angle: [145, 180]; kp_31_y_mean: [0.252, 0.336]	Full arm extension and elevated wrist (kp_31) maximize smash power and reach.

4 System Pipeline: From Raw Video to Pose Feedback

This section details the step-by-step pipeline implemented such that you have to just upload your training video and get a shot-type prediction and pose-diagnostic feedback at the end.

Step 0: Manual Frame Extraction

The pipeline begins with a raw match video uploaded to the working directory. Initially, I explored automating shot segmentation using **TrackNetV3** [3], which can be modified to identify

impact frames automatically based on motion of the shuttlecock. While promising, TrackNetV3 proved computationally expensive and infeasible to run repeatedly.

As a trade-off, I adopted a semi-manual alternative: visually scrubbing the video to identify impact moments and logging these frame numbers into a CSV file (`shot_frames.csv`). This balances speed and accuracy while allowing precise control over frame selection.

Step 1: Court Corner Detection Pipeline

With impact frames selected, I used the first frame of each shot to detect court geometry. The full pipeline for corner detection includes:

1. **Binary Masking**: Convert RGB frame to grayscale and apply adaptive thresholding to highlight court lines.
2. **Edge Detection**: Use Canny edge detection to isolate sharp linear transitions.
3. **Hough Line Transform**: Extract dominant lines corresponding to court boundaries.
4. **Line Filtering**: Discard short or near-horizontal/vertical lines that don't correspond to realistic court edges.
5. **Intersection Estimation**: Compute intersections of filtered lines to determine the four court corners (top-left, top-right, bottom-left, bottom-right).
6. **Corner Sorting**: Sort corners into a consistent order for homography computation.
7. **Verification**: Optionally visualize detected corners on the image to validate quality.

This pipeline operates entirely using classical vision techniques.

Step 2: Cropping to get Nearby Court

To ensure accurate keypoint extraction, I worked under the constraint that the player to be analyzed is positioned on the near side of the court. Using the court corners detected in the previous step, I computed the appropriate cropping box and saved the cropped frames accordingly.

Step 3: Homography Matrix Calculation and Application

Once the court corners are identified, I compute a **homography matrix** — a perspective transformation that maps the original camera view to a canonical top-down court view. This is done using:

```
H = cv2.getPerspectiveTransform(src_pts, dst_pts)
```

Here, `src_pts` are the detected court corners in the image, and `dst_pts` correspond to the idealized rectangular coordinates of the court.

Since the classification model and feedback system were trained using keypoints captured from a specific camera view (i.e., the training setup), I employ a two-step transformation during inference on new test videos:

1. Compute H_{test} to map keypoints from the test video's image coordinates to real-world court coordinates.
2. Apply H_{train}^{-1} to transform the real-world coordinates back into the image coordinate system of the training setup.

This ensures that all keypoints and spatial features extracted during testing are aligned with the same spatial reference frame used during training, making the model robust to different camera angles and recording conditions.

Step 4: Pose Keypoint Extraction

Using the **MediaPipe Pose** solution, I extracted 33 2D keypoints (`x, y, visibility`) from each cropped frame. This yields:

- A temporal sequence of body landmarks across the 61-frame shot window
- Skeleton annotations for upper/lower limbs, spine, and head

Step 5: Feature Computation

From the raw keypoints, I computed higher-order biomechanical features crucial for feedback:

- **Joint angles** (e.g., elbow, knee, shoulder)
- **Torso orientation** (e.g., lean angle, balance)
- **Vertical reach** (e.g., wrist above head)
- **Motion trends** (e.g., average hip movement)
- **Positional features** (e.g., court-relative foot position)

I also computed keypoint statistics such as `kp_0_x_mean`, `kp_0_y_max`, and `kp_24_y_std` to capture temporal and spatial significance. Each feature is computed per shot and aggregated into a final per-shot feature vector.

Step 6: Shot Classification Using Multi-Class Model

The system uses a multi-class classifier (XGBoost) to assign one of four shot types:

- Smash
- Drop
- Clear
- Net Shot

Step 7: Feedback Generation via Rule-Based Diagnostics

After shot classification, I generated pose-based feedback using a rule engine. Each shot type has associated threshold-based rules that determine optimality. For instance:

- **Smash**: If `elbow_angle < 120°` → "Extend elbow fully during impact."
- **Drop**: If `torso_lean_angle > 20°` → "Avoid excessive forward lean."

Step 8: Output Generation

The final output includes:

- A CSV file with per-shot predictions, computed features, and feedback
- Annotated impact frame images showing:
  - Predicted shot type
  - Feedback Highlights

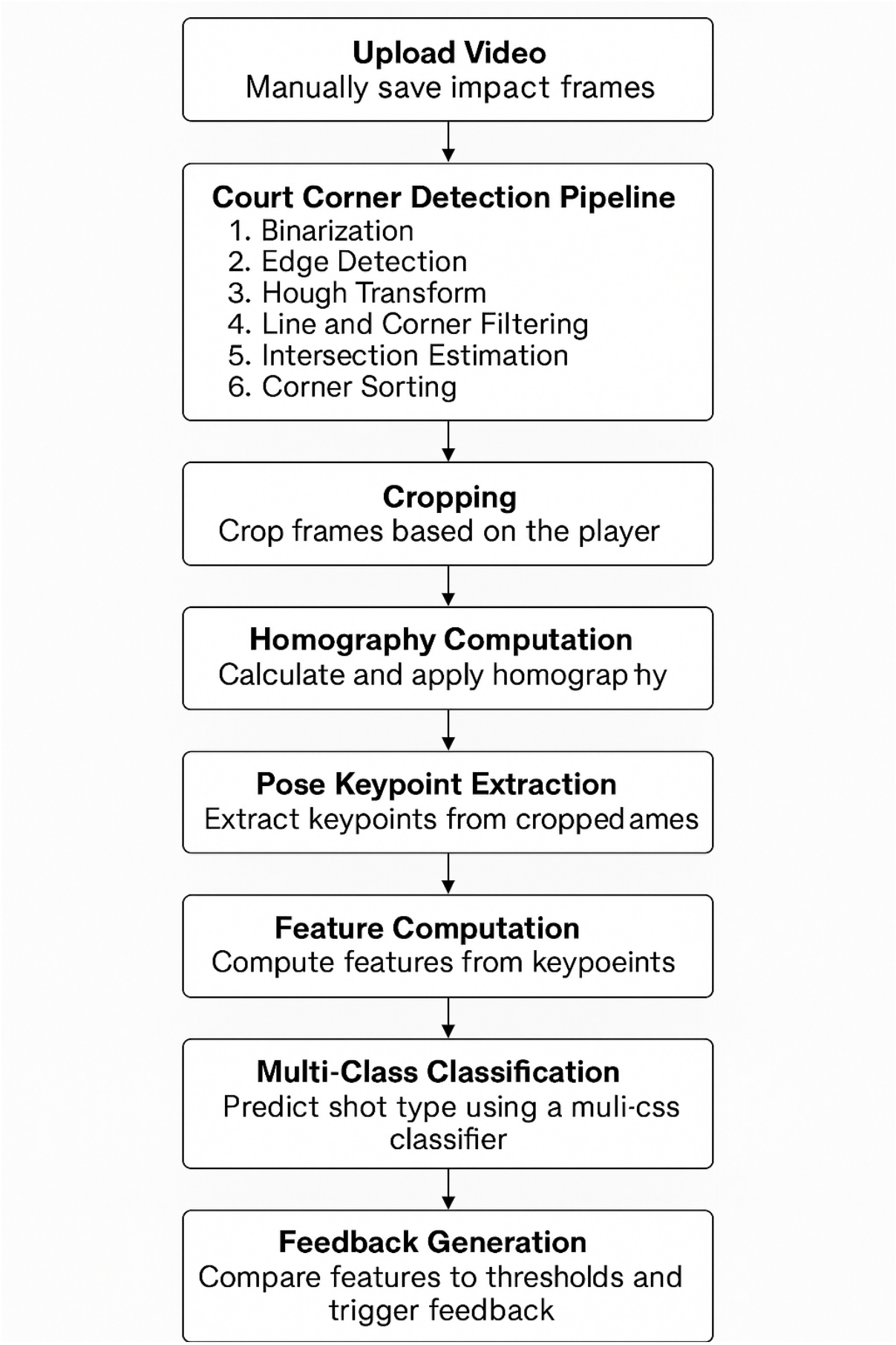

Figure 1: Flowchart of the entire pipeline from raw video to pose feedback.

Figure 2: Example of output: an annotated impact frame with predicted shot type and feedback highlights.

References

[1] ShuttleSet Dataset, CoachAI Project. https://github.com/wywyWang/CoachAI-Projects/tree/main/ShuttleSet

[2] *"Badminton Action Classification Based on PDDRNet"* by Xian-Wei Zhou, Le Ruan, Song-Sen Yu, Jian Lai, Zheng-Feng Li, and Wei-Tao Chen https://doi.org/10.2991/978-94-6463-230-9_118

[3] TrackNetV3: A spatiotemporal badminton rally parser. https://github.com/qaz812345/TrackNetV3