# A Multi-Camera Vision-Based System for Automated and Affordable Tennis Line Calling

GROUP PROJECT 3 – FINAL REPORT

**Luke Edwards, Joshua Featherstone, Echo Li, Sena Polat, Damian Stone**

{cm22683, qh22044, zu22178, qc22815, zo20876}@bristol.ac.uk

Supervised by Dimitris Agrafiotis

School of Electrical, Electronic and Mechanical Engineering

University of Bristol

2025

*This paper introduces a low-cost, vision-based electronic line calling (ELC) system for tennis that uses smartphones to achieve competitive accuracy. The system employs a hybrid object detection pipeline that combines YOLO, background subtraction, and optical flow to detect the tennis ball's 2D position using four iPhone cameras placed around the court. Stereo geometry is used to triangulate the ball in 3D space, and filtering and transformation techniques are used to refine the results.*

*A physics-based simulation framework based on Euler's method for projectile motion was developed to address the limited amount of data. Machine learning models were then trained using these simulations to accurately predict where the ball would land. A mean landing point error of 5.6 cm was attained by the final model, a Bi-Directional LSTM, alongside each predicted trajectory point having a confidence of 93.76%.*

# Declaration

This project report is submitted towards an application for a degree at the University of Bristol. The report is based upon independent work by the candidates. All contributions from others have been acknowledged and the supervisor is identified on the front page. The views expressed within the report are those of the authors and not of the University of Bristol.

We hereby assert our right to be identified as the authors of this report. We give permission to the University of Bristol Library to add this report to its stock and to make it available for consultation in the library, and for inter-library lending for use in another library. It may be copied in full or in part for any bona fide library or research worker on the understanding that users are made aware of their obligations under copyright legislation.

We hereby declare that the above statements are true.



Certification of ownership of the copyright in a dissertation presented as part of and in accordance with the requirements for the relevant degree at the University of Bristol.

# Work Allocation

The team's work allocation is further detailed on the contents page, where the contributors' initials are listed alongside the corresponding topics.

| Section | Student Contribution | | | | |
|---|---|---|---|---|---|
| | LE | JF | EL | SP | DS |
| Abstract | 60% | 10% | 10% | 10% | 10% |
| 1 | 50% | - | - | 50% | - |
| 2 | 10% | 60% | 10% | 10% | 10% |
| 3 | 20% | 20% | 20% | 20% | 20% |
| 4 | - | - | - | 100% | - |
| 5 | - | - | - | - | 100% |
| 6 | 80% | 10% | - | - | 10% |
| 7 | - | 100% | - | - | - |
| 8 | 10% | 60% | 10% | 10% | 10% |
| 9 | - | - | 100% | - | - |

| | | | |
|---|---|---|---|
| Luke Edwards | LE | Sena Polat | SP |
| Joshua Featherstone | JF | Damian Stone | DS |
| Echo Li | EL | | |

We confirm that the information on this page accurately describes our individual contributions to the project.

# Contents

# 1 Introduction

With a single line call determining the outcome of a tennis championship worth over £2 million [1], the importance of reliable, accurate, and accessible electronic line calling (ELC) systems cannot be overstated. Professional tennis tournaments commonly employ high-precision systems like Hawk-Eye, capable of accuracy within 2.6 mm [2]. However, the substantial cost of these solutions, estimated at $100,000 USD per court [3], places them beyond the financial reach of most clubs, amateur tournaments, and recreational players. Recognising the need for a more affordable yet similarly effective alternative, this paper introduces a multi-camera vision-based line calling system specifically designed around readily available smartphone cameras. Utilising four cameras strategically placed around the tennis court, our system processes multiple videos feeds and applies trajectory prediction algorithms to accurately determine where the ball hit the ground. This approach significantly lowers the barrier to entry, offering accuracy comparable to established commercial solutions at a fraction of the cost. By employing common, off-the-shelf hardware, our solution aims to democratise access to precise electronic line calling, thereby enhancing fairness, consistency, and accessibility in tennis at all levels.

## 1.1 Background (LE + SP)

The fastest tennis shot ever recorded was 163.7 mph [4]. As such, it should come as no surprise that experienced umpires cannot make line calls with perfect accuracy. A notorious example occurred at the 2004 US Open, where several incorrect line calls from the Auto-Ref system contributed to Serena Williams's loss [5]. While the chair umpire later overruled another clearly incorrect call, these errors sparked discussions about line-calling assistance - especially since the Auto-Ref system, then being tested by the U.S. Open, had otherwise shown high accuracy [6]. A major challenge for these systems is handling the speed of the tennis ball. Top players routinely hit serves and groundstrokes at speeds exceeding 120 mph. At such velocities, a ball can travel 0.8 m between the frames of a 60 fps camera, making it difficult to pinpoint the exact moment of contact with the court. The ball's contact with the ground lasts only a few milliseconds, and it often compresses and skids, blurring visual evidence of whether it touched the line. As a result, modern systems do not rely on a single frame to make close calls. Instead, they continuously track the ball and use this data to predict where it hits the ground between frames. Hawk-Eye, for example, uses ten high-speed cameras to record the ball's 3D trajectory and predict the precise landing spot on the court [7].

### 1.1.1 Existing Systems (LE)

Over the past two decades, several vision-based solutions have been developed. In 2006, Hawk-Eye was first deployed in professional tennis and quickly proved its value by virtually eliminating doubts on close calls. Originally, it had an error margin of about 4 mm (now ~2.6 mm), comfortably meeting the International Tennis Federation (ITF) requirement of ≤5 mm precision for electronic line judges [8]. Several other systems have entered the market since then, as shown in Figure 1.

| System | Accuracy | Hardware Requirements | Summary | Cost |
|---|---|---|---|---|
| Hawkeye | 3.6 mm (~5% the tennis ball's diameter) | >10 340fps Cameras | A professional-grade system used in major tournaments. It uses multiple high-speed cameras and advanced algorithms to track the ball's trajectory and determine line calls with high accuracy. | $100,000 |
| In/Out v4.0 | Millimetre accuracy | 2 Net Device v4.0 + 2 Line Devices | A portable consumer device featuring two cameras on a net post and two at the baseline. It offers millimetre-accurate line calls and provides instant audio/visual "in" or "out" signals. Designed for everyday players seeking professional-like precision at a low cost. | $897 |
| Baseline Vision | 97% accuracy for calls within 10 cm of the line | 1 Set of 60fps Stereo Cameras | A mid-tier solution that uses visual analysis to determine line calls. It strikes a balance between affordability and performance, suitable for clubs and competitive amateurs. | $1,999 |
| FOXTENN | 100% accuracy within 5 mm of the line | >40 2,500fps Cameras + Laser System | Uses a high-speed laser scanner system and more than 40 ultra high-speed cameras, generating over 100,000 images per second. Captures every movement and bounce on the court with extreme accuracy and reliability. Primarily used in professional-level tournaments. | $50,000 per week |

*Figure 1: Summary of existing line-calling solutions* [9] [10] [11] [12]

### 1.1.2 Recent Academic Approaches (SP)

While these commercial systems have set the standard for tennis ELC, recent applications in academic literature highlight an increased interest in accessible, open-source alternatives.

Wong [13] demonstrated a promising approach with a total cost of $200 USD and precision of 8mm, using four webcams positioned around the court. This solution applied intrinsic and extrinsic calibration methods that resulted in a reprojection error of 3.78 pixels, background subtraction for the ball detection and tracking, and triangulation with average error of 1.00 pixel.

Fazio et al. [14] further advanced the field with a system that processes smartphone videos for 3D trajectory estimation of a tennis ball in play. This solution utilised image processing concepts including segmentation, morphological operations, the Hough transform, moments and blob analysis, sensor calibration, and multiple-view geometry.

These promising papers highlight the potential and challenges of developing an affordable tennis ELC system. A critical limitation across existing approaches is the lack of robust and reliable of ball detection. Our solution addresses this gap by demonstrating a hybrid approach leveraging cutting-edge technology.

## 1.2 Project Objectives (LE)

The aim of this project is to develop a low-cost, automated line calling system that meets competitive accuracy standards using only consumer-grade devices. The specific objectives are:

1. Identify cost-effective and accessible sensors that provide optimal results.
2. Detect and track the tennis ball accurately from video footage.
3. Triangulate the ball's 3D position using the 2D video coordinates and camera geometry.
4. Predict the exact contact point on the court to meet ITF officiating standards.
5. Estimate the ball's flight and trajectory.
6. Make reliable line calls and maintain accurate game scoring.

Stretch Goals

7. Explore the potential for generating real-time match statistics.

Through these objectives, the project aims to demonstrate that a highly accurate and affordable tennis line-calling system is achievable. By using multiple camera viewpoints to simulate stereo vision, along with modern computer vision techniques, a smartphone-based setup could rival the performance of professional systems like Hawkeye. The following sections of this report outline the system design, implementation, and evaluation of the proposed solution.

## 1.3 System Design (LE)

Based on the project objectives, we have divided the problem into three key components and four supporting sub-components, as shown in Figure 2. These elements work together to form a complete pipeline that captures match footage, triangulates the ball's position in 3D space, models its trajectory, and determines the outcome of each shot.

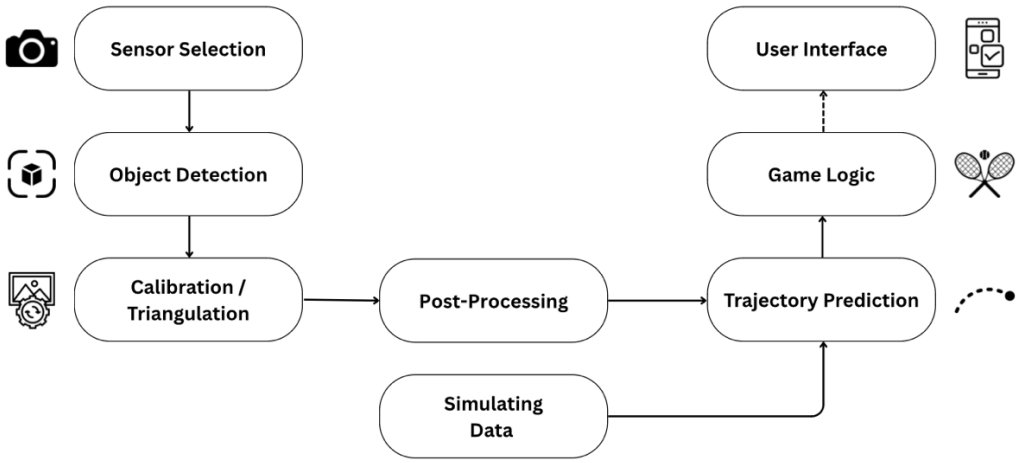



*Figure 2: Overview of project workflow*

## 2 Hardware Considerations

### 2.1 Sensor Arrangements Considered (JF)

As we scale the number of sensors by two, our data volume scales similarly by the same factor. This exponential scaling implicitly implies we cannot adopt an approach with an exceptionally high number of sensors. Throughout this section, the trade-off between processing load and locational accuracy must be considered in parallel. Keeping the above logic in mind, the potential number of cameras is narrowed to three main arrangement scenarios highlighted below in Figure 3.

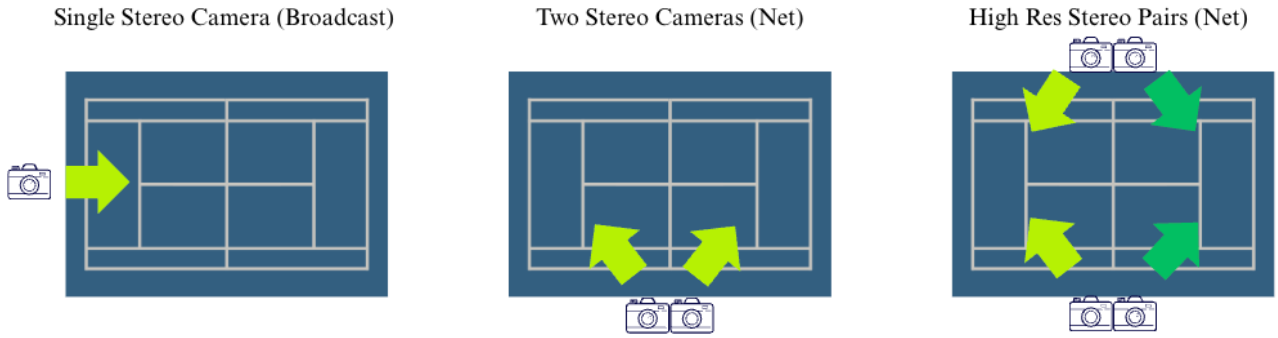



*Figure 3: Stereo and non-Stereo camera arrangements*

For future reference, the maximum distance from the camera required for a line call to be considered IN for Net and Broadcast positions is displayed in the Figure 4 below. These are 27.6m and 16.8m, respectively.

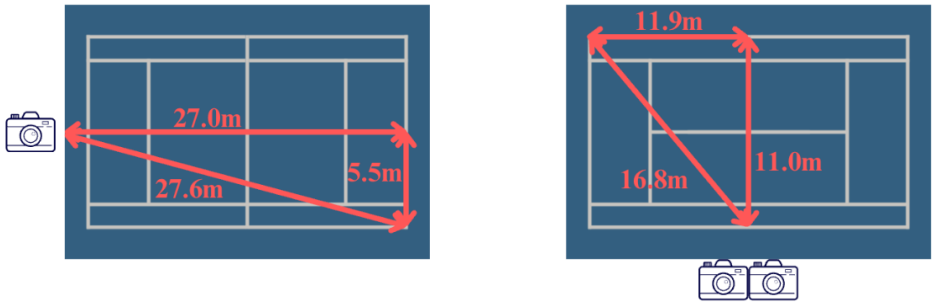



*Figure 4: Net and broadcast maximum distances*

### 2.2 Stereo Sensor Discussion (JF)

Stereo cameras have their intrinsic, built-in depth fields that are calculated by internal triangulation. They return an overlayed depth measurement for every 2D point in the camera's plane [15]. These are often displayed as heat maps as shown in Figure 5, with the colour gradient mapping to depths over its operating ranges. For stereo cameras the depth measurement is crucial to accurately place the ball in 3D space. Using a single stereo in place of two ordinary cameras would remove the need for post processing (triangulation) and lessen the overall processing load. Initial checks were essential to ensure the depth would have enough accuracy

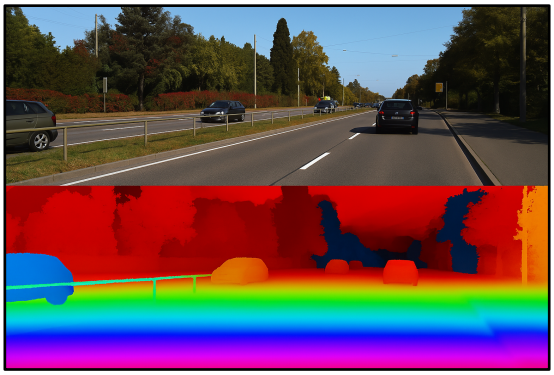


*Figure 5: Stereo overlay*

## 2.3 Single Stereo (JF)

For a single stereo camera, as shown in Figure 6, the maximum depth was 27.6m (1.d.p), assuming the camera is set on a tripod 2m back from the baseline. We set out to find a stereo solution that operated successfully within this depth range. After conducting research for the best single stereo camera, we landed on the Luxonis Oak-D LR [16] as the best possible option. This camera has 60 fps and its operating ranges up to 30m appeared ideal. However, it cost $699 and required additional lens attachments to improve its FOV from the baseline 82 degrees. Even at the optimum 40-degree FOV, we would have approximately 4% depth error for our maximum 27.6m distance, equating to errors of order 1m in just that measurement alone, discounting the single stereo approach

| HFOV [°] | <2% error | <4% error | <8% error |
|---|---|---|---|
| 40 | 13.2 m | 33.0 m | 65.9 m |
| 82 | 5.5 m | 13.8 m | 27.6 m |

*Figure 6: Luxonis Oak-D LR depth errors*

## 2.4 Dual Stereo (JF)

Similarly, for a dual stereo camera the maximum depth is instead 16.8m (1.d.p), assuming the camera is set up on a tripod level with the net post. After conducting research for the best dual stereo camera setup, we landed on a pair of the Luxonis Oak-D S2 [17]. This has up to 120 fps and operating ranges up to 12m. Below highlighted is a comparison of different options available.

| Camera | Price | Global Shutter | FOV (170° Needed) | Frame Rate | Stereo | Built-in AI | Distance | Connect |
|---|---|---|---|---|---|---|---|---|
| Luxonis OAK-D S2 PoE | $399 | YES | 2 x 89.5 | 120 FPS | YES | YES | IDEAL 70cm - 12m | PoE |
| Luxonis OAK-D S2 | $249 | YES | 2 x 89.5 | 120 FPS | YES | YES | IDEAL 70cm - 12m | USB |
| Stereolabs ZED 2 | $449 | NO | 2 x 110 | 100 FPS (NATIVE) | YES | YES | IDEAL 0.2m - 20m | USB |
| Gemini 335L | $359 | YES | 2 x 90 | 60 FPS | YES | YES | IDEAL 25cm - 6m | USB |

*Figure 7: Stereo camera options*

However, again the depth error of 6% for our maximum distance still leaves an unacceptable depth uncertainty region of orders of 0.5m. The below depth data is cited directly from the Luxonis website. For reference, Figure 8 also shows a 1m and 0.5m uncertainty range around an example 6cm diameter ball. This clearly demonstrates stereo depth sensing has far below the adequate capabilities, leaving us with the remaining option of four cameras.

| Distance Range | Depth Accuracy |
|---|---|
| < 4 m | < 2% absolute error |
| 4 m - 7 m | < 4% absolute error |
| 7 m - 10 m | < 6% absolute error |

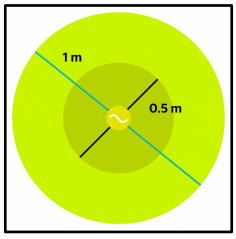



*Figure 8: Luxonis Oak-D S2 depth errors and visualisation*

## 2.5 2D Error Factors (JF)

This then left only the final and most rigorous option of 4 individual 2D cameras. The triangulation of the pair of 2D co-ordinates in each stereo pair (shown in the final diagram in Figure 3), mapping onto the global 3D co-ordinate system, uses pure trigonometry and doesn't introduce additional error. Therefore, the only error related conversation while selecting sensors, was whether the theoretical 2D error factors within a single frame are within sensible margins, which would be carried through the triangulation process. The combined error placing a tennis ball in a 2D frame is the combined quantisation and motion blur errors [18], both summarised in the equation below [19].

$$\epsilon_{total} = \sqrt{\epsilon_q^2 + \epsilon_m^2} \quad \epsilon_m = \nu \cdot t \quad \varepsilon_q = \frac{\left(2 \cdot D_x \cdot \tan\left(FOV/2\right)\right)}{\text{Resolution width}} \tag{1}$$

The motion blur effects are product of the ball velocity and shutter time. Selecting a high shutter speed of 1/1000 for the maximum ball speed of 67 m/s gives a motion blur of 0.067 m or 6.7 cm. Average 48 m/s speed results in a motion blur of 4.8 cm, within reasonable bounds as of the approximate order of the ball diameter. Looking now at quantisation error, how objects with further depth relative to the sensor take up less pixels in frame than those closer. Using the cited equation below, pixel size at the maximum distance of 16.8 meters with a 60° field of view and a resolution of 1280 pixels (720P) is approximately 0.0152 meters per pixel or 1.5cm per pixel. This results in a quantisation error of about 0.75 cm. The combined 2D error factors are therefore acceptable. For any further calculations the 2D error on placing the ball location is assumed 7cm.

## 3 Data Collection (LE)

After investigating the potential sensors and configurations in Section 2 and researching our methods for capturing and triangulating the ball in Sections 3 and 4, we chose to use four identical iPhone cameras mounted 1.4m above the ground, as shown in Figure 9.

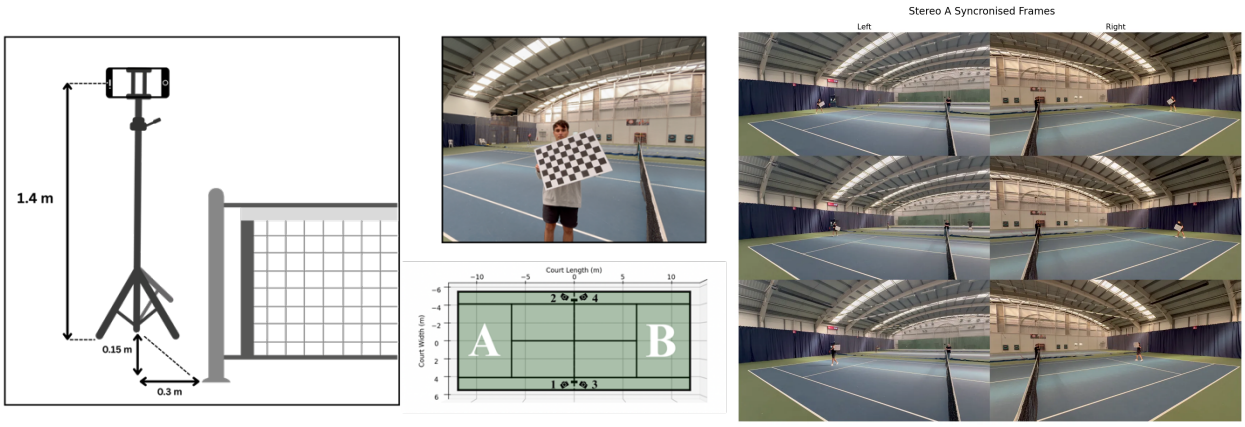



*Figure 9: Data collection*

We used this setup to collect footage over two sessions. In the first session, we tested multiple camera angles and positions to identify flaws in our approach, guiding our method for collecting the final test footage.

# 4 Tennis Ball Detection

## 4.1 Overview of Object Detection and Tracking Techniques (SP)

Reliable and accurate detection and tracking of a tennis ball has presented significant challenges in Electronic Line Calling (ELC) solutions. While the ITF standard requires accuracy within 5mm, achieving this is complicated by several factors: the ball may be occluded by players and the net, and moves at extremely high speeds, with professional serves averaging 120 mph [20]. Additionally, the ball's appearance varies drastically due to motion, lighting conditions, background activity, video compression, and the presence of other tennis balls. To address these challenges, we implemented a hybrid solution that leverages the strengths of classical and cutting-edge computer vision techniques.

### 4.1.1 YOLO (SP)

YOLO (You Only Look Once) is a state-of-the-art, open-source object detection framework maintained by Ultralytics. It directly processes global images in a single pass through a convolutional neural network for an accurate and computationally efficient object detection [21].

One can either custom train a YOLO model or use the generalised out-of-the-box models, which are trained on COCO data and contain a variety of labels. The output includes the class of the detection, as well as confidence score and bounding box. Each version and sub-version of YOLO have quantitative metrics to measure and compare performance, such as the mAP50 and mAP50-95 (mean average precision) scores and inference time. These values can drastically improve following preliminary and fine-tuning training.


*Figure 10: Demonstration of YOLO object detection*

### 4.1.2 Background Subtraction KNN with Morphological Operations (SP)

Background subtraction is a widely used classical computer vision technique for isolating the foreground which contains moving objects, by comparing each frame to a learned 'background' model. This method is particularly applicable to tennis ball tracking given the static cameras and court, and the ball's distinctive size and movement. There exist many algorithms implementing background subtraction, such as the Mixed-Gaussian or Consensus approaches. However, our solution employs a K-Nearest-Neighbour (KNN) algorithm for background/foreground segmentation. This is to leverage its relatively low computational cost and robustness to noise and illumination changes [22]. OpenCV's KNN Background Subtractor maintains a history of the N previous frames to classify whether a new pixel belongs to the foreground or background. This classification depends on when the Euclidean distance between it and the K-nearest samples in the background model differ significantly from the normal [23]. This adaptive approach handles small foreground objects and lighting differences effectively [24].

### 4.1.3 Lucas-Kanade Optical Flow (SP)

The Lucas-Kanade optical flow algorithm has been a simple, robust, and reliable method of object tracking and path mapping since 1981 [25]. Optical flow is a computer vision tool for describing the motion of objects in video sequences via a vector field. This matrix describes the displacement of each pixel between two consecutive frames [26]. The assumption leading to this algorithm is the brightness constancy constraint, which states that the brightness, $I$, of a point at location $(x,y)$, having moved by $\Delta x$, $\Delta y$ in a short interval $\Delta t$, would remain the same, i.e.:

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t) \tag{2}$$

Lucas-Kanade assumes that optical flow is constant within a small window of the frame. Hence, the optical flow equation holds for all pixels within this window, creating a system of equations that can be solved using the least squares method. The algorithm handles the aperture problem (where motion along edges is ambiguous) by considering this neighbourhood of pixels and finding the local image flow vector $(V_x, V_y)$ that minimises the sum of squared differences between frames. This makes it particularly effective for tracking tennis balls, especially when features are selected at high-contrast points on the ball's edge. However, Lucas-Kanade requires a key point input, which can either be selected using Shitomasi corner detection [27] or a previous detected centre of the ball.

### 4.1.4 HSV Colour Segmentation and Hough Transform Circle Detection (SP)

The Hough transform [28] is a feature selection method, which transfers the image from the spatial domain to the parameter space, that can detect where a circle is most likely to be with high accuracy. For tennis ball detection, we can alter the circle radius range and create strong contrast between the ball and background.

To enhance the effectiveness of the Hough Transform circle detection, the background must be eliminated, as it is likely for the algorithm to detect circles elsewhere in the image. This can be done by implementing segmentation based on the pixel values in the Hue-Saturation-Value (HSV) colour space. We can manually define the HSV range within which the tennis ball can be and then perform thresholding operations to obtain only the pixels per frame which satisfy the range. From here, we can perform an 'Opening' morphological operation, which erodes the boundaries of the foreground objects before dilating them to counteract the pixel loss. This reduces noise which increases the likelihood of successful Circle Detection as shown in Figure 11.

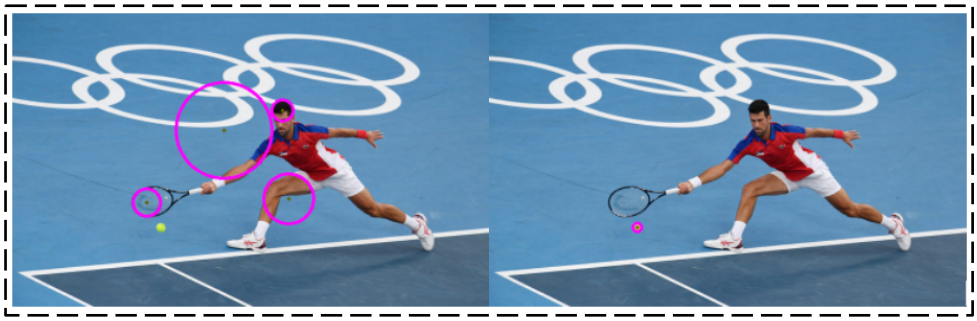


*Figure 11: Hough Transform before and after HSV segmentation and radius constraint* [29]

## 4.2 Detection of the Ball in Video Frames (SP)

### 4.2.1 Hybrid Detection (SP)

The hybrid solution developed in this paper leverages the strengths of the multiple methods with several fall-back processes, visualised in Figure 12. In short, YOLO and Background Subtraction validate each other, and, if neither yield a valid detection, the system falls-back to Lucas-Kanade tracking or HSV segmented Hough circle detection. This decision flow was devised following several iterations to maximise successful detection rates and reduce average inference time.

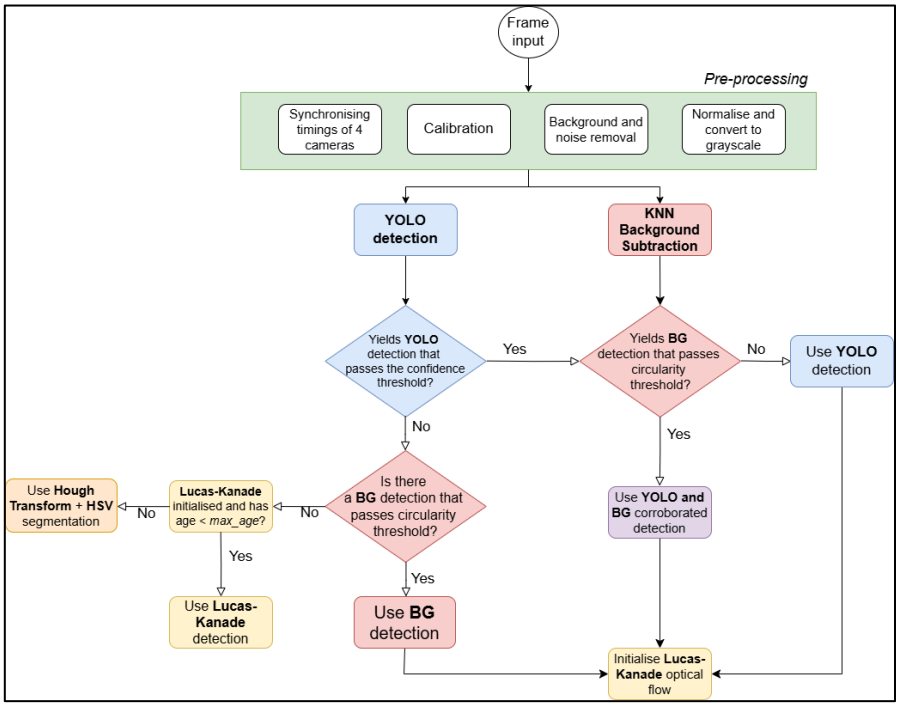



*Figure 12: Decision flow of hybrid system per frame*

When the system yields a successful detection for a frame, the detection is appended to an array of other detections for further processing before being input to the triangulation section of the pipelines.

### 4.2.2 Algorithms and Implementation (SP)

Several computer vision techniques and mathematical algorithms were implemented for this hybrid system to mitigate ball detection challenges in affordable ELC.

**Background Subtraction Implementation**

Our implementation of background subtraction and motion extraction addresses the complication of separating the ball in play with moving players with big object removal. The following algorithm, demonstrated in Figure 13, describes this process for a given frame [30].

1. Background subtraction with a threshold produces a map of pixels, $M_b$, with objects
2. Closing morphological operations (dilation then erosion) performed on $M_b$ to fill the gaps within objects (the players)
3. Employ big object removal to remove players from $M_b$ to obtain $M2_b$
4. Calculating the difference between consecutive frames with a threshold yields a map, $M_d$, of pixels with moving objects
5. Calculate a refined map $M_t = and(M2_b, M_d)$
6. Assuming the ball is in motion, pixels around $M_t$ from previous frame cannot be the ball's position. Calculate a map, $M_{pre}$, to clean up those pixels around the player not detected by big object removal
7. The final map with the tennis ball will be $M_f = and(M_t, M_{pre})$

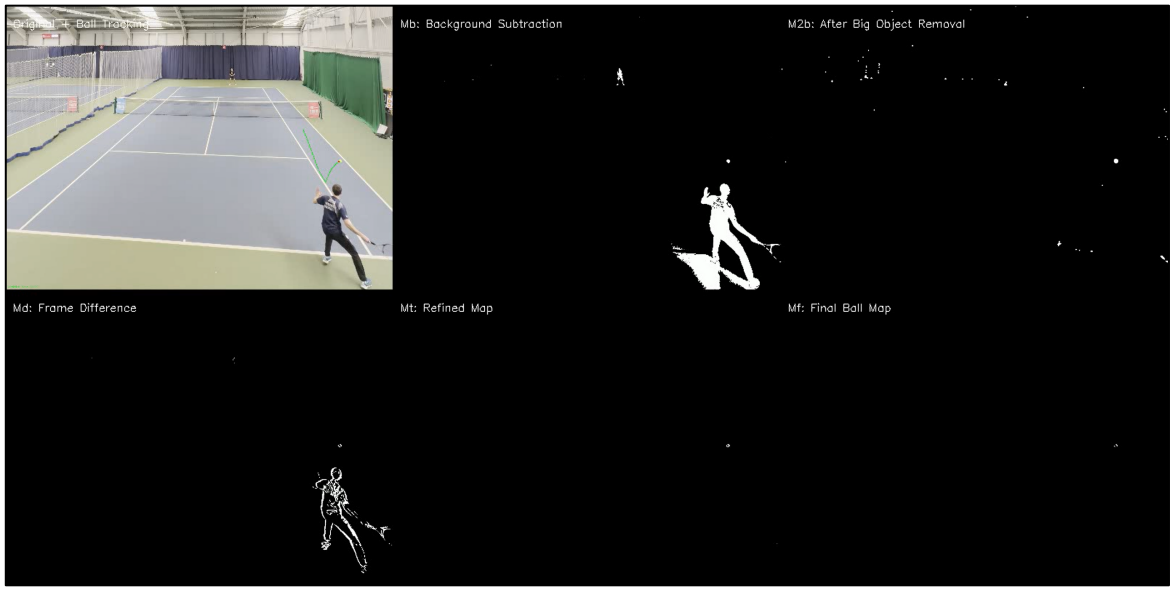



*Figure 13: The pixel maps used in background subtraction for ball tracking*

This algorithm with background subtraction yields a robust and high performing ball detection method, given its tolerance to lighting changes, distance from the ball and to the effects of motion (for instance, blur and squash).

For reliable ball candidate selection, we analyse contour properties, particularly circularity, which is calculated using the common shape compactness ratio:

$$circularity = 4\pi \cdot \frac{area}{perimeter^2} \quad (3)$$

A perfect circle has a circularity measure of 1.0, and the tennis ball typically yields values above 0.5 despite motion blur, squash, and other distortions.

**YOLO Detection Implementation**

The hybrid system's usage of the custom-trained YOLO model is enabled by the Ultralyrics-maintained Python library. Through this library, we instantiate our model and call the predict function to return a list of object detections describing the class, confidence level, and bounding box (i.e. the minimum enclosing box) of the detection. The library also enables us to test and validate the model with any set of labelled images.

### Lucas-Kanade Optical Flow Implementation

Our optical flow implementation employs the pyramidal version of Lucas-Kanade, implemented by the OpenCV library. The pyramidal implementation iteratively builds an image pyramid with multiple resolution levels, consequently handling larger motion between frames, while keeping a small window of integration and achieving high local accuracy [31]. Each camera object in the object-oriented system has an attribute *lk_max*, which defines the maximum 'age' allowed for consecutive Lucas-Kanade detections without being re-initialised. This maximum was introduced to minimise the instances where faulty or blurry detections have led the Lucas-Kanade key point to latch onto the background or another object, leading to limitless false positive detections.

### 4.2.3 YOLO Model Training + Fine-Tuning (SP)

Improving the performance of the YOLO object detection involved iterative fine-tuning and creating and labelling our own dataset. The initial two training iterations utilised pre-existing datasets from Roboflow in YOLOv5 and v8 formats, which provided a foundation for tennis ball detection and performed well on professional footage [32]. However, they lacked the specific characteristics for our testing environment.

For the final and most successful iteration, we created a custom dataset by extracting random frames from our actual test videos captured with consumer-grade cameras. This process involved manually labelling 369 frames to precisely identify tennis balls under exact lighting conditions, camera angles, and motion patterns present in our system. Figure 14 demonstrates the improvements in metrics after 100 epochs of training. This custom dataset was essential for optimising detection performance for our specific conditions.

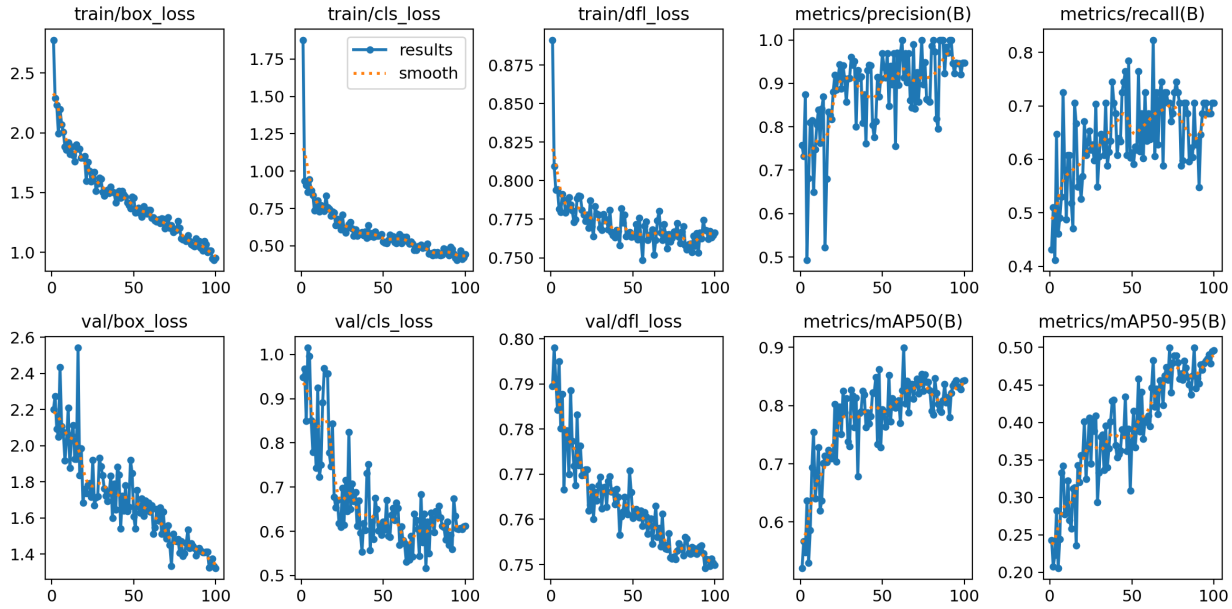



*Figure 14: Training improvement metrics following the final YOLO fine-tuning iteration against number of epochs*

We implemented a split of 78% for training, 14% for validation, and 8% for testing. This distribution was chosen to balance several considerations. The validation set was slightly enlarged, with some of the images duplicated with ±20% lighting variation, to provide more robust hyperparameter tuning during training. This prevents the model from over-fitting, which would yield a high proportion of false positives. This data split configuration proved effective, with the final model achieving high performance metrics while maintaining good generalisation capabilities as shown in Figure 15.

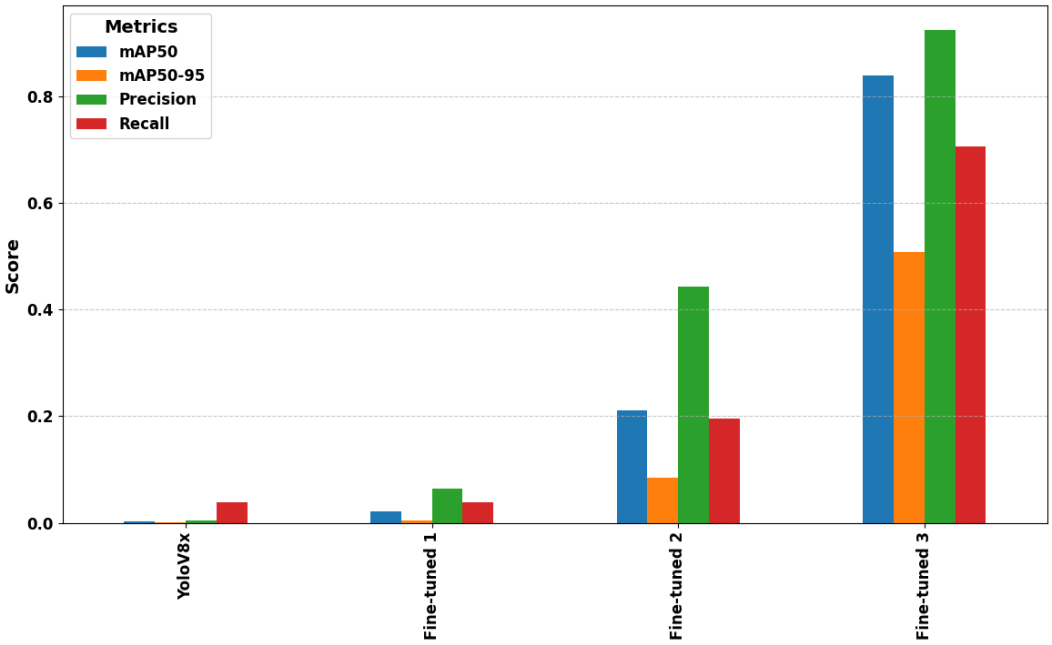



*Figure 15: Performance comparison after fine-tuning iterations [NVIDIA GeForce RTX 3060 Laptop GPU]*

This comparison shows a 42050% increase in mAP50 score, 23575% increase in precision, 1710% increase in recall, and a more than 10ms decrease in inference time between the out-of-the-box YOLO V8x model and the final fine-tuned model. However, this comparison is only reflective of the applications to our videos given the validation and testing set; more fine-tuning and a larger validation set would lead to more robust models and more reflective validation metrics.

## 4.3 Validation of Detection Methods (SP)

### 4.3.1 Quantitative Performance Metrics (SP)

We can validate the performance of our hybrid system via triangulation; comparing the performance of different methods assesses the credibility of our final system. To obtain comparison metrics, we ran the individual computer vision techniques, the individual techniques supplemented with Lucas-Kanade, and the Hybrid system on 50 consecutive frames from one of the stereo angles. This allowed for quantitative measurements of successful detection rate, inference time (time taken to make a detection), and the maximum and average of the consecutive frames without a successful detection, exhibited in Figure 16.

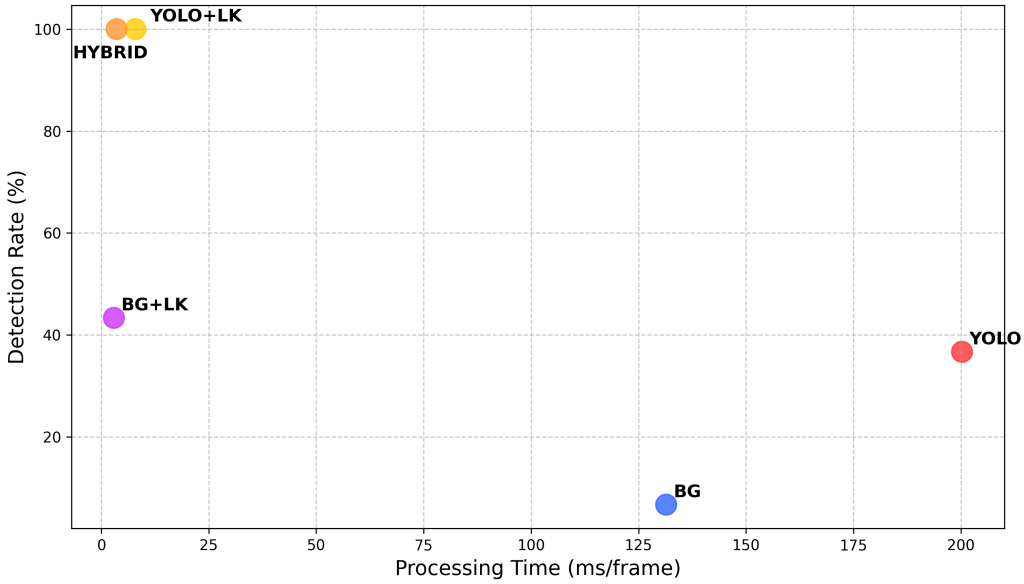



*Figure 16: Detection rate vs inference time of different object detection implementations*

The quantitative comparison indicates that the Hybrid system has the highest rate of successful detections as well as the shortest inference time. As demonstrated in Figure 17, YOLO augmented with Lucas-Kanade optical flow yielded a similar detection rate and inference time. However, visualising all the detections simultaneously for the frame range showed that YOLO and LK resulted in several consecutive periods of false positives, due to the optical flow not being reinitialised as frequently as the Hybrid system. One limitation is that detection rate does not always reflect accuracy.

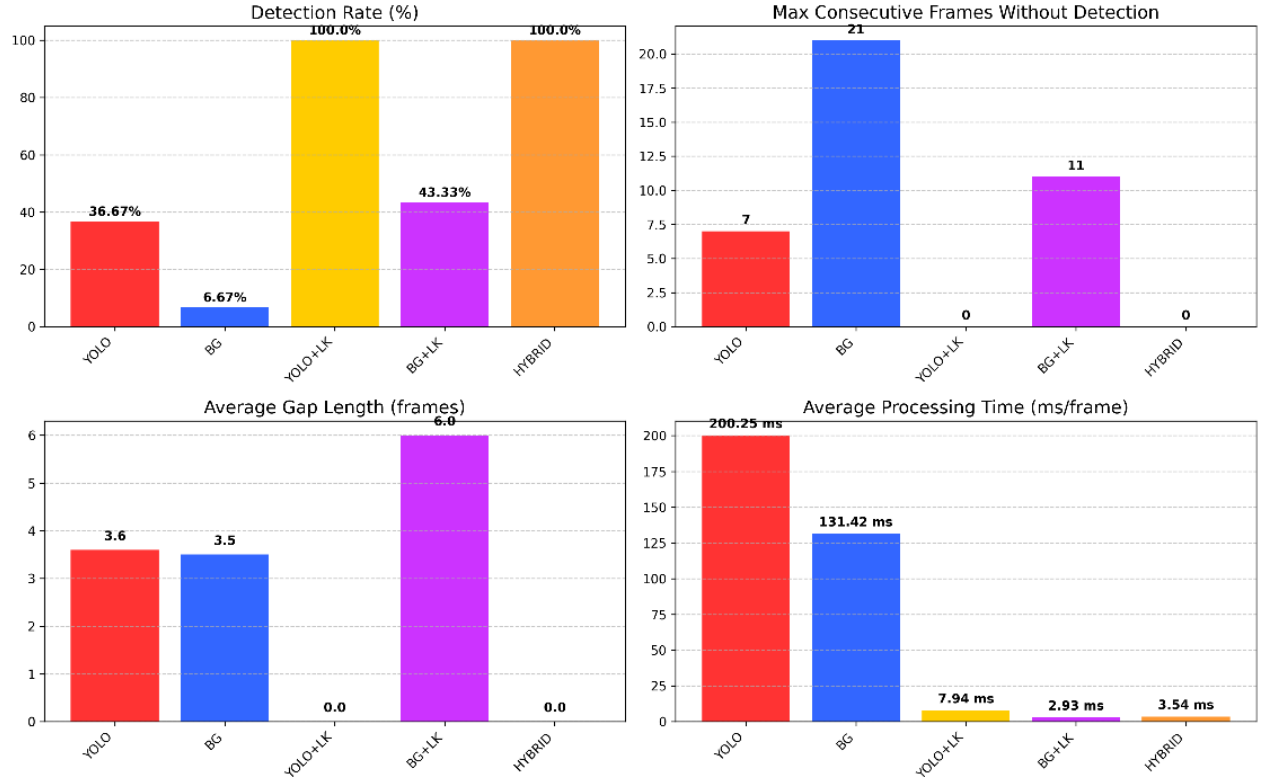



*Figure 17: Comparison of object detection methods on consumer-grade footage*

### 4.3.2 Challenges in Object Detection (SP)

The challenges in object detection lie at every stage, from the sensors to the detection methods. Our testing revealed that even ball motion at moderate speeds causes distortion; the 60 frames per second resolution of the sensors can make the ball appear as an elongated streak, extending 3-4 times the ball's diameter. Test videos at 120 fps in slow-motion yielded 4% higher successful detection of the ball.

Lighting in a non-professional setting is a prominent challenge, as it causes low contrast between the ball and the background, and the yellow colour of the ball to not be visible. This affects YOLO, Background Subtraction, and HSV segmentation as detection methods. This can be mitigated by some pre-processing and integrating trajectory prediction, as well as the physical placement of the sensors. Sensors at a higher position looking down at the court will be less affected by background lights.

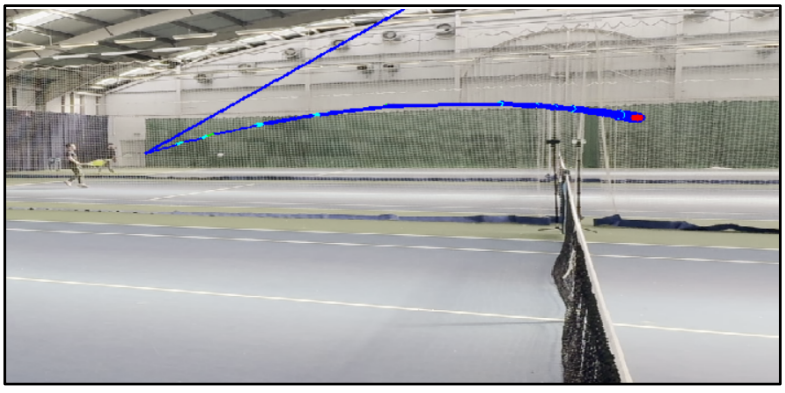


*Figure 18: Background subtraction detecting the ball-in-play in the neighbouring court*

A challenge arising from the sensor positioning was the system detecting balls in play on neighbouring courts, exemplified in Figure 18, or static balls which were not in play. While these detections were not technically false positives, they were problematic for the triangulation and 3D point extraction. Aside from sensor placement, the effect of this could be reduced with additional post-processing and handling of points, for example not recording detections that are far from the previous detection or while there are validated stereo detections on the opposite court. Almost all of these detections could be filtered by either confidence level, lack of detection from both stereo cameras, or anomalous error observed during post-processing in Section 5.7.

### 4.4 Integration with Triangulation (SP)

The object detection pipeline was designed to prepare data for triangulation by producing synchronised detections across stereo pairs. We developed a specialised data structure and CSV export format for the detection output to facilitate the triangulation process, ensuring consistent frame matching, shown in Figure 19.

This output dataset contains the detections from all 4 cameras per frame, with the detected centre of the ball, confidence score, and method features included. This enabled debugging, continuous development, and for the triangulation algorithm to weigh the reliability of detections by confidence score.

The detection system underwent multiple iterations to optimise its output format and detection thresholds to minimise false positives while maintaining high recall.

| frame_no | cam1_x | cam1_y | cam1_confidence | cam2_x | cam2_y | cam2_confidence | cam3_x | cam3_y | cam3_confidence | cam4_x | cam4_y | cam4_confidence |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1048 | 1507 | 408 | 0.827627 | 442 | 406 | 0.816876 | 16 | 374 | 0.797673 | 1892 | 383 | 0.300000 |
| 1049 | 1545 | 410 | 0.805354 | 441 | 406 | 0.400000 | 71 | 383 | 0.832680 | 1745 | 404 | 0.848748 |
| 1050 | 1541 | 413 | 0.400000 | 373 | 410 | 0.634832 | 60 | 382 | 0.400000 | 1700 | 411 | 0.832230 |
| 1051 | 1541 | 413 | 0.300000 | 372 | 409 | 0.400000 | 178 | 402 | 0.845032 | 1657 | 417 | 0.833924 |
| 1052 | 1666 | 421 | 0.833444 | 311 | 437 | 0.658793 | 225 | 410 | 0.840000 | 1614 | 425 | 0.757897 |
| 1053 | 1710 | 426 | 0.668895 | 310 | 437 | 0.400000 | 248 | 411 | 0.735438 | 1575 | 431 | 0.803263 |
| 1054 | 1755 | 432 | 0.832334 | 207 | 430 | 0.832046 | 311 | 425 | 0.761611 | 1575 | 423 | 0.400000 |
| 1055 | 1755 | 428 | 0.400000 | 161 | 436 | 0.838825 | 353 | 434 | 0.835638 | 1575 | 423 | 0.300000 |
| 1056 | 1795 | 434 | 0.711496 | 113 | 442 | 0.400000 | 353 | 433 | 0.400000 | 1486 | 447 | 0.618625 |
| 1057 | 1873 | 448 | 0.814763 | 62 | 451 | 0.855797 | 425 | 451 | 0.844724 | 1431 | 459 | 0.825900 |

*Figure 19: Hybrid detection output (input of triangulation pipeline)*

### 4.5 Future Improvements (SP)

Future iterations of the object detection system would address all of the challenges in Section 4.3.2 and provide better performance and more features as a cutting-edge tennis line calling solution.

A novel algorithm would remove any false positive detections or balls not in play – this could also be implemented by integrating the detection with the trajectory prediction and implementing more comprehensive edge case handling. The integration with trajectory prediction would provide a feedback loop in which the predictions constrain the search space for ball detection. This could result in an improvement of successful detections by more than 18% [Section 5.7], leading to more accurate 3D reconstruction and increased number of sample points to determine line calls.

A further object detection improvement which would enable generation of enhanced match statistics would be introducing player detection. This could be achieved by applying state-of-the-art pose estimation technology such as SkeleTRACK [33] or Swin Poseformer [34], or by developing our bespoke algorithm by extracting semantic key points.

Furthermore, while the hybrid system reduced the inference time from just YOLO detection by more than 98% in some cases, the inference time could be reduced further by implementing hardware acceleration through GPU optimisation. This would increase processing speed by an estimated 60-70% [35], approaching real-time processing and reducing the bottleneck that object detection causes in the pipeline.

These enhancements would be validated using a more comprehensive test set designed to improve performance in challenging conditions, including occlusions, lighting and contrast variation, and multiple balls in frame. These improvements would benefit the overall line-calling system performance.

# 5 Calibration and 3D Ball Localisation

## 5.1 Calibration of Stereo Cameras Overview (DS)

Stereo camera calibration is the process of determining the internal (intrinsic) parameters of each camera and position and rotation of one camera to another (extrinsic parameters). This results in a mathematical model that enabling depth estimation and mapping 2D image points to 3D.

The main goal of stereo calibration is to obtain accurate intrinsic, extrinsic, and distortion parameters that explain how one camera views the world from the other's perspective. If these parameters are accurate, this will lead to precise triangulation, which computes the 3D position of the ball.

This process works by capturing a common object with known dimensions (often a chessboard) to provide fixed and realisable 3D reference points. Each chessboard corner's true position is known, and its corresponding 2D projection in each camera image is detected. These correspondence points are then used to solve for the following pinhole camera model equation [36].

$$s \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \cdot [R \mid T] \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{4}$$

Where:

- $s$: scaling factor.
- $u, v$: horizontal and vertical pixel coordinates 2D points.
- $K$: intrinsic parameters.
- $R, T$: extrinsic parameters.

By passing many correspondence points of the known object to this equation, then we can apply a nonlinear optimisation, such as Levenberg-Marquardt, to estimate the final intrinsic and extrinsic parameters [37]. This optimisation will minimise the error between the observed and the projected 2D points computed from the 3D points using the calibrated camera parameters, also known as the reprojection error.

### 5.1.1 Intrinsic, Extrinsic & Distortion Parameters (DS)

Intrinsic parameters define each camera's internal optical characteristics, determining how 3D points project onto a 2D plane. These parameters include focal lengths ($fx$, $fy$) and optical centre ($cx$, $cy$), expressed in a 3×3 camera matrix K which maps 3D points to 2D coordinates under the pinhole camera model.

$$[K] = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

Extrinsic parameters describe the spatial relationship between two cameras, specifically their rotation and translation relative to each other. These are represented by a rotation matrix ($R$) and translation vector ($t$), forming a transformation that aligns both cameras in the same 3D coordinate system, as shown in Figure 20.

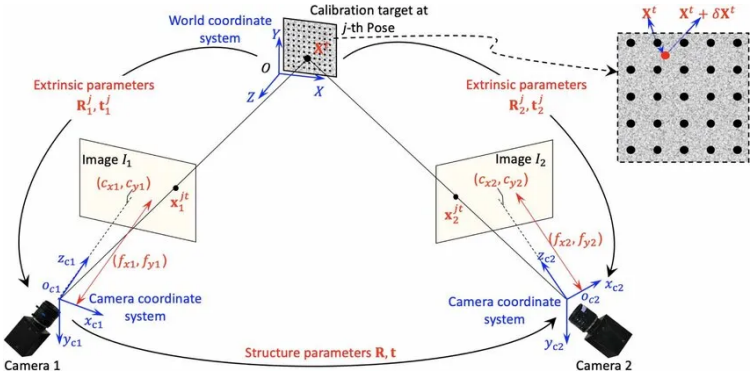



*Figure 20: Extrinsic parameters defining the spatial relationship between two cameras* [38]

Intrinsic calibration also involves distortion parameters, demonstrated in Figure 21; real lenses introduce imperfections like radial distortion (curved straight lines) and tangential distortion (lens-sensor misalignment). During calibration, distortion coefficients are estimated to compute an undistorted camera matrix, providing a more accurate view by removing lens effects. This leads to better extrinsic parameter computation and improved 3D reconstruction accuracy.

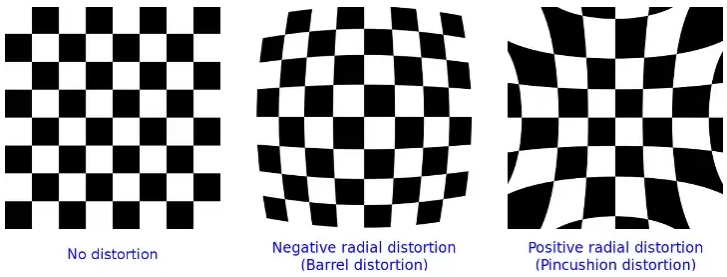



*Figure 21: Common lens distortion in camera calibration* [39]

### 5.1.2 Calibration Methods (DS)

During the literature review, we explored methods for calibrating stereo cameras in the context of ball tracking on sports fields, focusing on tennis courts.

For intrinsics, we found two options. First, using manufacturer-provided parameters like focal length and distortion coefficients. Second, the more common approach in academic and practical setups: capturing multiple images of a chessboard (or planar pattern) from different angles. This is simple and fully supported in OpenCV via the `calibrateCamera` function [40].

To obtain the extrinsic parameters, both cameras must capture a shared, measurable object to compute rotation and translation. We explored two main approaches:

- **Using court lines:** Using tennis court lines as reference, as in Farin et al., 2004 [41], we can extract extrinsic parameters by detecting line intersections and comparing them to a known model.

- **Using a shared chessboard:** Placing a chessboard in both camera views and capturing multiple images. Known 3D geometry allows calculation of camera rotation and translation by minimising reprojection error, supported by OpenCV's `stereoCalibrate` function [42].

Since we used 0.5x wide-angle lenses on consumer-grade cameras, we could not rely on manufacturer intrinsics as they were not available. Therefore, we decided to calibrate intrinsics using the chessboard method, capturing multiple angles and positions for each camera.

For extrinsic, we first considered court-line calibration [43] but our limited field of view and overlapping lines from adjacent courts made this method too sensitive to noise. However, as the system runs offline, we decided to calibrate each stereo pair independently and later align both pairs using rigid transformations [44]. Given these constraints, we also selected the chessboard method for extrinsic, ensuring reliable calibration by capturing the board from multiple angles in both cameras simultaneously for each stereo.

## 5.2 Calibration Methodology (DS)

To carry out calibration for this project, we separated the intrinsic and extrinsic parameter-refinement processes into independent pipelines, enabling focused improvement of each.

To gather data for these processes, we recorded:

- A video from each camera showing the chessboard at various angles to cover the full field of view.
- A synchronised stereo video for each pair (stereo A and B), with the chessboard visible in the overlapping view of both cameras.

### 5.2.1 Intrinsic Calibration Pipeline (DS)

After extracting frames from the videos, we obtained approximately 50 frames per camera, covering the full view with the chessboard in various positions. The pipeline works as follows:

1. Each frame's chessboard corner positions are detected using `cv.findChessboardCorners` and then refined to sub-pixel accuracy for precision.
2. Intrinsic parameters are estimated from all corner points using `cv.calibrateCamera`.
3. A `compute_reprojection_error()` function projects the 3D points back into the image with the initial parameters via `cv.projectPoints`, yielding a score, where lower values indicate better calibration.
4. Frames with high reprojection error are filtered out using a threshold to improve accuracy; as long as more than 10 frames remain, `cv.calibrateCamera` is run again on the filtered set.
5. The distorted camera matrix is computed from the distortion parameters using `cv.getOptimalNewCameraMatrix.`
6. Finally, the results for each camera were saved as a JSON file for future extrinsic and triangulation calculations.

### 5.2.2 Extrinsic Calibration Pipeline (DS)

After calibrating each camera's intrinsics, we process stereo image pairs of the same chessboard pose (left and right). The pipeline proceeds as follows:

1. For each stereo pair, the left and right frames are loaded, then the chessboard corners are detected in both using `cv.findChessboardConers` and `cv.cornerSubPix.`
2. The reprojection error for each image is computed using `cv.solvePnP` to verify the reliability of corner detections, filtering out noisy samples.
3. A reprojection threshold is applied to ensure both images have good detection and low error, retaining valid object and image points.
4. Once valid pairs are collected, the previously calculated intrinsic parameters are used with `cv.stereoCalibrate` to estimate the extrinsic parameters: rotation matrix (R) and translation vector (T), which define the relative position of the right camera to the left one. Finally, the results are saved into a JSON file for future triangulation calculations.

## 5.3 Calibration Improvements (DS)

Reprojection error measures the distance between observed 2D image points and their projected positions, computed from corresponding 3D points using the estimated camera parameters. In this case, the points are the chessboard corners. Lower error means more accurate camera settings and better 3D reconstruction. The standard acceptable reprojection error is below 1, ideally as small as possible, while maintaining a large and varied set of frames with different positions.

Firstly, for both intrinsic and extrinsic, the function `detect_chessboard` was optimised, adding settings such as `cv.CALIB_CB_ADAPTIVE_THRESH` and `cv.CALIB_CB_NORMALIZE_IMAGE` to handle lighting changes and make the chessboard easier to detect.

**Algorithm 1** Detect Chessboard Corners

**Require:** $frame$, $pattern\_size$, $win\_size \leftarrow (5, 5)$
**Ensure:** $ret$, $corners$
$gray \leftarrow$ cvtColor($frame$, COLOR_BGR2GRAY)
$flags \leftarrow$ CALIB_CB_ADAPTIVE_THRESH $\vee$ CALIB_CB_NORMALIZE_IMAGE
$(ret, corners) \leftarrow$ findChessboardCorners($gray$, $pattern\_size$, $flags$)
**if** $ret$ **then**
    $criteria \leftarrow$ (TERM_CRITERIA_EPS + TERM_CRITERIA_MAX_ITER, 50, 0.001)
    $corners \leftarrow$ cornerSubPix($gray$, $corners$, $win\_size$, $(-1, -1)$, $criteria$)
**end if**
**return** $(ret, corners)$

*Figure 22: Corners Detection Function using OpenCV*

We improved corner detection by adjusting the settings of `cv.cornerSubPix`, which refines the positions of the corners. We increased the number of iterations per image and reduced the window size, allowing it to

focus more on detecting smaller corners, which was important for distant chessboard positions where the squares appear tiny. The difference between accurate and inaccurate chessboard corner detection is demonstrated in the following Figure 23.

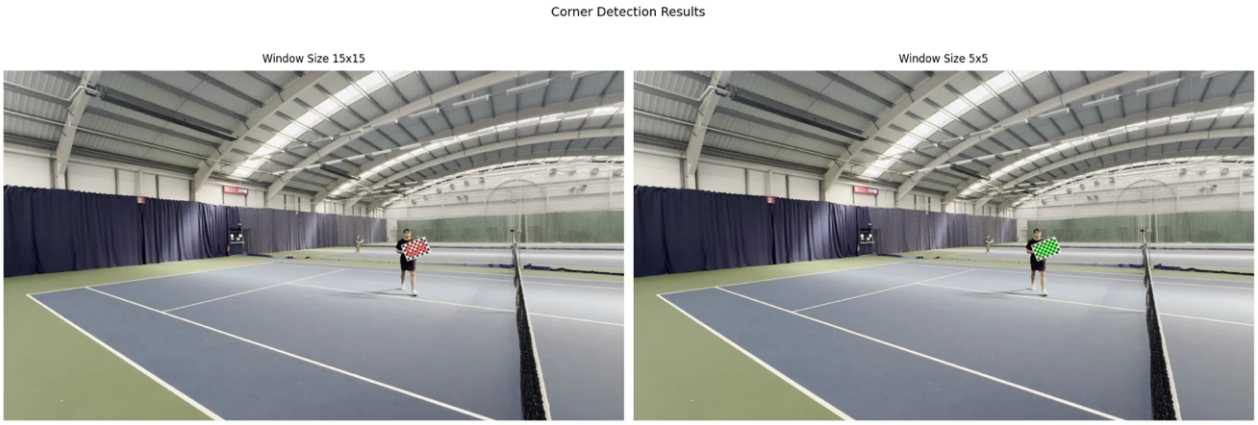



*Figure 23:* Left: larger window struggles refining small corners. Right: smaller window yields more accurate corners

## 5.4 Intrinsic Improvements (DS)

To determine the best settings for intrinsic calibration, we first tested different window sizes, i.e. the region around a detected chessboard corner where the algorithm searches for the precise subpixel position. As exemplified in the left plot of Figure 24, as the window size decreases, the reprojection error improves. In this case, we identified 5x5 as the best window size before applying any frame filtering.

Secondly, a reprojection error threshold was applied to filter out low-quality frames, focusing on high-quality, diverse examples. As shown in the right plot of Figure 24, testing different thresholds demonstrates that removing low-quality frames improves the reprojection error.

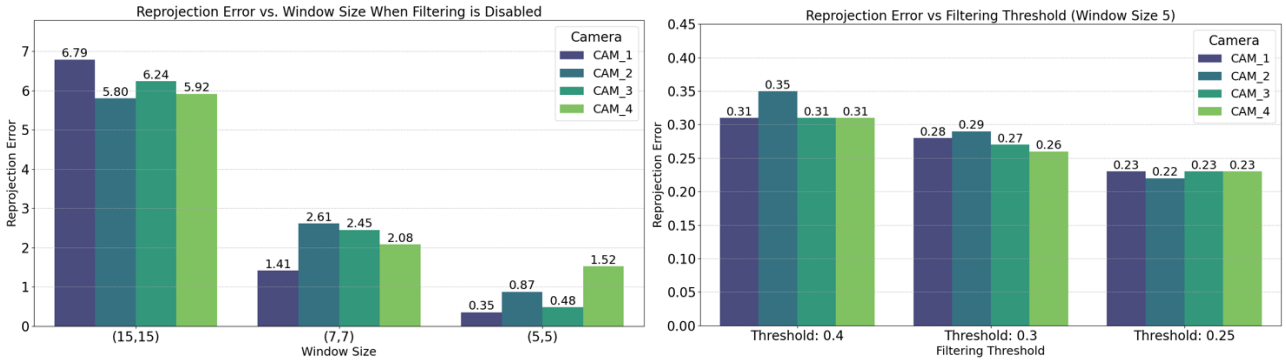



*Figure 24: Effect of window size and frame filtering on reprojection error*

To select the best threshold for our calibration, as shown in the Figure 25, we considered the reduction in frames, as it is important to retain as many frames as possible that cover the entire camera view, with the chessboard in different positions, for a robust calibration.

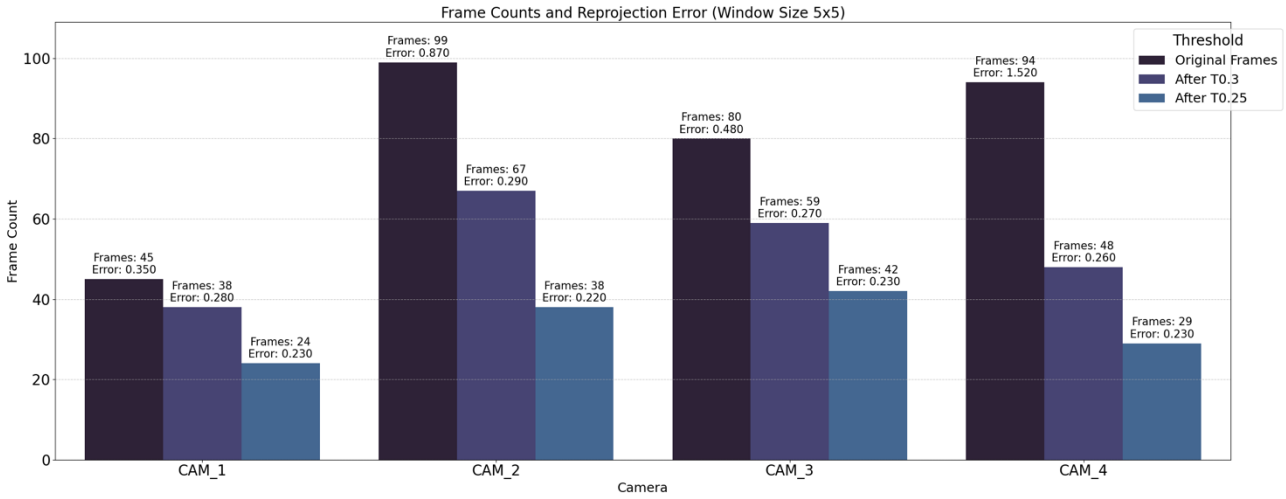



*Figure 25: Frame count vs. reprojection error across filtering thresholds*

In this case, as shown in Figure 25, a threshold of 0.3 was selected over 0.25. Although the reprojection error is slightly lower with threshold 0.25, the difference is minimal, and threshold 0.3 retains significantly more frames, providing a better balance between error and dataset size. Lower thresholds resulted in fewer than 10 frames, so they were not considered.

| Camera | Frames Before | Reproj. Error Before | Frames After | Reproj. Error After | Frames Reduction % | Error Reduction % |
|---|---|---|---|---|---|---|
| CAM_1 | 45 | 0.3500 | 38 | 0.2800 | 15.56% | 20.00% |
| CAM_2 | 99 | 0.8700 | 67 | 0.2900 | 32.32% | 66.67% |
| CAM_3 | 80 | 0.4800 | 59 | 0.2700 | 26.25% | 43.75% |
| CAM_4 | 94 | 1.5200 | 48 | 0.2600 | 48.94% | 82.89% |

*Figure 26: Final intrinsic calibration results using 5×5 window and threshold 0.3*

### 5.4.1 Extrinsic Improvements (DS)

For extrinsic calibration shown in the following Figure 27, we followed a similar chessboard-based approach, therefore the same window size was used (5x5). Consequently, the different thresholds were tested to reduce the initial reprojection error, aiming for a value below 1.0.

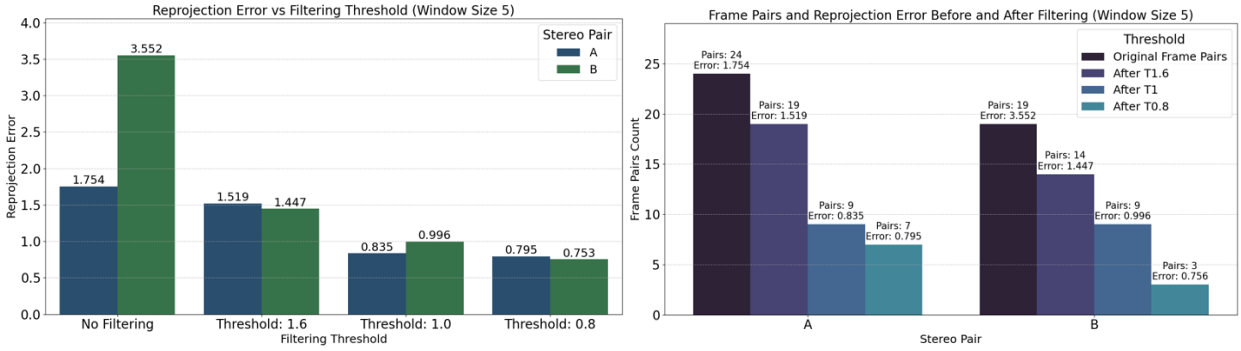



*Figure 27: Reprojection error by threshold (left) and frame pair impact (right)*

Unfortunately, the number of initial frames available for extrinsic calibration was limited, restricting further filtering and the ability to select more accurate frames. Nevertheless, we achieved a reprojection error below 1.0, using thresholds 1.0 and 0.8. We chose threshold 1.0 to retain more stereo frame pairs, prioritising a realistic calibration over a slight decrease in reprojection error. In summary, the final configurations achieved a 52.41 % reduction in reprojection error for Stereo A and a 71.95 % reduction for Stereo B.

| Stereo Pair | Frame Pairs Before | Reproj. Error Before | Frame Pairs After | Reproj. Error After | Frames Reduction % | Error Reduction % |
|---|---|---|---|---|---|---|
| Stereo A | 24 | 1.7538 | 9 | 0.8347 | 62.50% | 52.41% |
| Stereo B | 19 | 3.5519 | 9 | 0.9964 | 52.63% | 71.95% |

*Figure 28: Final extrinsic calibration results using 5×5 window and threshold 1.0*

### 5.5 Triangulation Overview (extracting 3D points) (DS)

Triangulation is the process of determining a 3D point by intersecting projection rays from two or more calibrated cameras [45]. Using known intrinsic and extrinsic parameters, along with corresponding 2D image points, it calculates the true 3D location of objects from different viewpoints.

In this project, the main goal of triangulation is to convert 2D ball detections into real-world 3D coordinates. By applying the calibration parameters, the 2D points detected by the hybrid algorithm are processed to compute the ball's true 3D position. The result is two sets of 3D coordinates in pixels, one for each stereo pair (A and B). These are later converted to metres, then merged and aligned during the trajectory post-processing step.

#### 5.5.1 How Triangulation Works

Once both cameras are calibrated, we can use their projection matrices to find the 3D coordinates of the tennis ball from its 2D detections. The projection matrices for a stereo pair are typically defined as:

$$P_1 = K_1[I \mid \vec{0}] \tag{6}$$

$$P_2 = K_2[R \mid t] \tag{7}$$

Where:

- $K_1, K_2$ are the intrinsic matrices of cameras 1 and 2, respectively
- $I$ is the 3x3 identity matrix (indicating camera 1 is at the origin)
- $O$ is the zero-translation vector for camera 1
- $R$ and $T$ describe the rotation and translation that position camera 2 relative to camera 1

We use the first camera of each stereo as a reference, making it the origin (0,0,0) for triangulation. This way all 3D points are computed relative to this fixed reference, which simplifies the alignment of points across different stereo pairs.

When a point is observed at pixel coordinates $x_1, x_2$ in both cameras:

$$x_1 = [u_1, v_1, 1]^T \tag{8}$$

$$x_2 = [u_2, v_2, 1]^T \tag{9}$$

The triangulation problem seeks the 3D point $X$ that satisfies:

$$\hat{x}_1 \sim P_1 X \tag{10}$$

$$\hat{x}_2 \sim P_2 X \tag{11}$$

Where $\hat{x}_1$, $\hat{x}_2$ are undistorted image points and $X$ is the 3D point in homogeneous coordinates.

As shown in Figure 29, calibration imperfections prevent the two rays from intersecting perfectly.

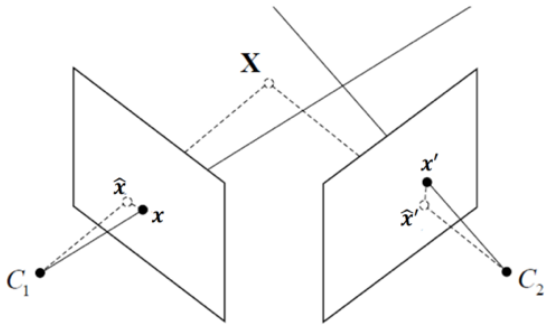



*Figure 29: Triangulation error caused by imperfect camera calibration* [46]

Therefore, it is possible to use methods like Direct Linear Transform (DLT) [47] to convert these equations into a linear system $AX = 0$. The DLT algorithms solves for $X$ selecting the solution that minimises the total reprojection error. The result is the most geometrically consistent 3D point given both camera observations. This step can be easily computed using `cv.triangulatePoints()` from OpenCV [48].

### 5.6 Triangulation Methodology (integration with Object Detection) (DS)

As previously mentioned, this stage aims to convert 2D ball detections into 3D spatial coordinates through triangulation. As shown in Figure 19, the input data consists of hybrid-YOLO detections, where each camera provides the ball's (x, y) coordinates in pixels. These stereo pairs are defined as **Stereo A (**Cameras 1 and 2) and **Stereo B** (Cameras 3 and 4).

#### 5.6.1 Triangulation Pipeline

We developed a Python pipeline to process and validate 2D ball detections from two independent stereo rigs (Stereo A and Stereo B). Each pair was processed independently to simplify debugging and avoid unnecessary recalibration when only one stereo setup had issues.

This design also made it easier to validate each post-processing step and any required transformations.

1. **Load calibration & compute projections -** We loaded each stereo pair's intrinsic and extrinsic parameters to compute it's projection matrices. The world coordinate system was aligned to camera 1 (for Stereo A) and camera 3 (for Stereo B), making that camera the origin (0, 0, 0) for the corresponding stereo. All calculations were performed using the undistorted camera matrices.
2. **Scale detections -** The hybrid algorithm's ball detections come from a 1080p video frames, whereas the intrinsic and extrinsic calibration was performed using chessboard images captured at 4k resolution. To reconcile this difference, a scale factor of 2 was applied to each object detection (x, y) coordinate before further processing, ensuring compatibility with the calibrated camera parameters.
3. **Undistort points -** The wide-angle lenses we use introduced barrel (radial) distortion in the raw detections. After scaling, we corrected the 2D points with `cv.undistortPoints`, which maps distorted pixel coordinates into a normalised, undistorted image-plane using the undistorted camera matrix and distortion coefficients.
4. **Triangulate to 3D -** Using the undistorted 2D points from both cameras, `cv.triangulatePoints` was applied to compute the homogeneous 3D coordinates of the detected object.
5. **Convert to Cartesian -** The homogeneous output was converted into Cartesian coordinates (X, Y, Z) by normalising with respect to the homogeneous component.
6. **Validate via reprojection** - Finally, to assess the triangulation accuracy, the 3D coordinates were reprojected back into 2D space using the original projection matrices. The Euclidean distance between these reprojected points and the original 2D detections was computed, providing a pixel-wise error metric.

**Algorithm 1:** Triangulate and Validate Stereo Detections

**Input :** `path_out`, `scale_factor`, `pixel_thresh`, `conf_thresh`, `detections_csv`, `matched_frames_dir`, `reproj_frames_dir`, $F_x^{(1)}, K_1, D_1, F_x^{(2)}, K_2, D_2, R_{\text{stereo}}, t_{\text{stereo}}$

**Output:** `valid_count`, `avg_err_cm_cam1`, `avg_err_cm_cam2`

$(P_1, P_2) \leftarrow$ getProjectionMatrices$(K_1, K_2, R_{\text{stereo}}, t_{\text{stereo}})$; `// 0. Compute camera projection matrices`

$(csv, n_{\text{after}}, n_{\text{prev}}) \leftarrow$ preprocessDetections$(detections_csv, conf_thresh)$; `// 1. Preprocess detections`

valid_count $\leftarrow 0$;

**foreach** *row* ***in*** *csv* **do**

  print("Frame", row.frame_no)

  $p1 \leftarrow (row.A1_x \cdot scale, row.A1_y \cdot scale)$; `// 2a. Scale and undistort 2D detections`

  $p2 \leftarrow (row.A2_x \cdot scale, row.A2_y \cdot scale)$;

  $u1 \leftarrow$ undistortPoints$(p1, K_1, D_1)$ $u2 \leftarrow$ undistortPoints$(p2, K_2, D_2)$;

  $X_h \leftarrow$ triangulatePoints$(P_1, P_2, u1, u2)$; `// 2b. Triangulate to 3D`

  $X \leftarrow X_h[1:3] \,/\, X_h[4]$

  $(e1, e2, \hat{p}1, \hat{p}2) \leftarrow$ computeReprojError$(X, P_1, P_2, u1, u2)$; `// 2c. Reproject and compute error`

  $(e1_{cm}, e2_{cm}) \leftarrow$ toCentimeters$(e1, e2, F_x^{(1)}, F_x^{(2)}, X)$

  **if** $e1 > pixel\_thresh \vee e2 > pixel\_thresh$ **then**

    anomalyDetected←true; `// 2d. Flag anomalies + tally valid`

  **else**

    anomalyDetected←false;

    valid_count++

  appendToCSV(path_out,frame=row.frame_no,$X, e1_{cm}, e2_{cm}, anomalyDetected$)

$(avg1, avg2) \leftarrow$ getAverageErrorCM$(csv)$; `// 3. Compute overall avg error`

**return** *valid_count*, *avg1*, *avg2*

*Figure 30: Triangulation and reprojection validation pipeline*

### 5.7 Data Accuracy and Refinement of 3D Points (DS)

This stage focused on improving the accuracy of triangulated points, with particular attention to the $x$ and $y$ reprojected coordinates. Additionally, the goal was to capture and structure the necessary data for post-processing and trajectory analysis, ensuring a smooth transition into the next stage of the project.

Videos and ball detections were taken at 60 fps, meaning there were many detections per second. For trajectory analysis, quality matters more than quantity, so the first step was filtering out low-confidence (≤ 20%) detections to remove noisy data.

Once filtered, the next focus was identifying and removing anomalies from the triangulated points. Anomalies were defined as cases where the reprojected 2D points were more than 30 cm away from the original object detections. The conversion from pixel error to centimetres was done using the $\text{Error}_{\text{cm}}$, formula, which allowed us to apply a real-world threshold for reliable anomaly detection.

$$\text{Error}_{\text{cm}} = \frac{\text{Error}_{\text{cm}} \cdot \text{Z}}{\text{fx} \cdot 10} \tag{12}$$

This method was especially effective at catching three common cases:

- Hybrid detection algorithm falsely detected balls from nearby courts, which would still produce high confidence scores but result in triangulated points far from the true location.
- One of the camera pair produced a wrong detection, pulling the triangulated point away from the true ball position.
- Calibration errors, especially from poorly estimated intrinsic or extrinsic parameters, caused systematic shifts in triangulation.

At this point, the process exposed an important insight: many valid ball detections were being incorrectly flagged as anomalies. To investigate, we visualised the reprojected points on the original ball detection frames.

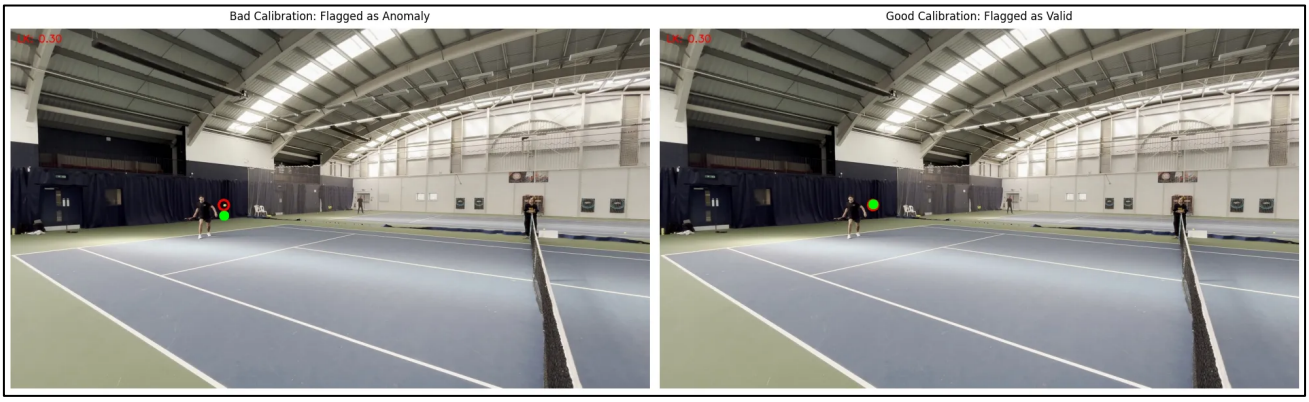



*Figure 31: Inaccurate vs accurate calibration of the same triangulated point (green dot)*

This step confirmed the root cause was not the hybrid object detection algorithm, but calibration errors, especially in stereo B. Poor calibration shifted triangulated points, making valid detections look like anomalies. (see Figure 31).

Figure 32 shows this clearly: with poor calibration, anomalies increase, and valid reprojected points are far off from the original detections. With refined calibration, the reprojected points align closely to the originals, confirming less centimetre error. In Figure 32, poor calibration (top) results in scattered red points, while refined calibration (bottom) shows tight overlap between red (reprojected) and blue (original) points.

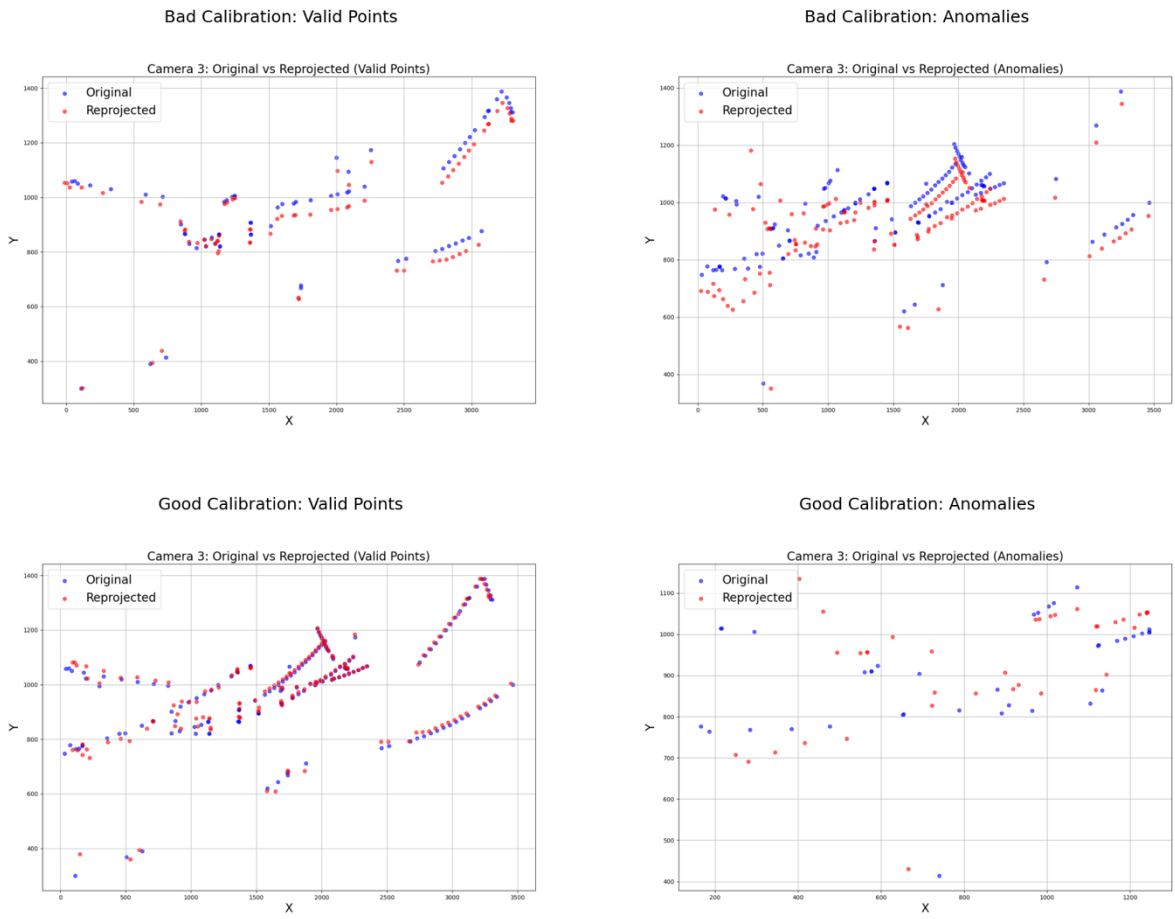



*Figure 32: Inaccurate vs. accurate calibration of reprojected coordinates vs original detection coordinates*

To solve this in stereo B, we refined the intrinsic and extrinsic parameters using a more diverse and varied set of chessboard images. After recalibration, the average reprojection error dropped significantly, and we achieved an 80% increase in valid detections, dropping average centimetre error from more than 12 to less than 5 centimetres. (see *Figure 33*).

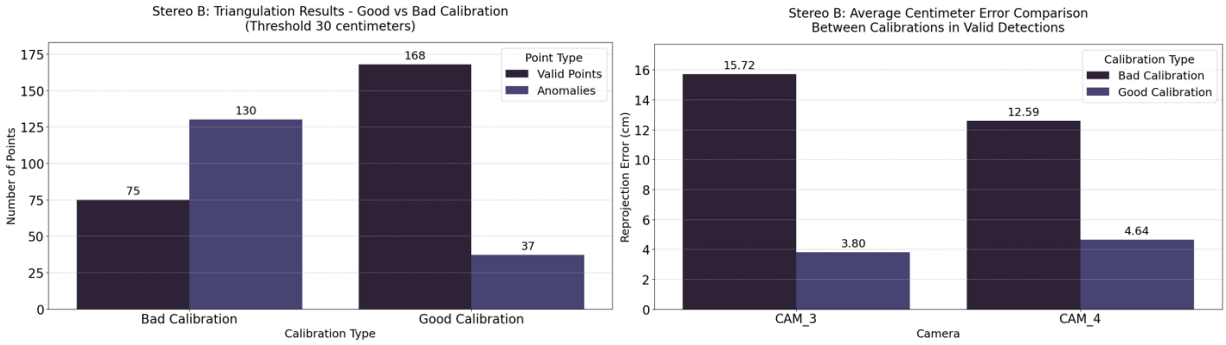



*Figure 33: Valid vs anomalous detections under inaccurate and accurate calibration (left) and centimetre error per camera comparison (right)*

Finally, Figure 34 displays the results after processing anomalies with improved calibration for both Stereo A and Stereo B. The final outcomes show an average error of 5.11 centimetres for Stereo A and 4.22 centimetres for Stereo B, indicating a high level of accuracy in reprojecting the ball's position relative to its original location.

| Stereo Pair | Raw Points | Filtered Points | Valid Points | Anomalies | Valid % | Anomalies % | Avg Error (cm) |
|---|---|---|---|---|---|---|---|
| Stereo A | 317 | 225 | 207 | 18 | 92.00% | 8.00% | 5.11 |
| Stereo B | 320 | 205 | 168 | 37 | 81.95% | 18.05% | 4.22 |

*Figure 34: Final triangulation results after post-processing and calibration correction*

# 6 Coordinate System Registration and Trajectory Normalisation

## 6.1 Triangulation Output Analysis and Initial Challenges (LE)

After completing the triangulation in Section 4, we are left with a series of coordinates and frame indices for each stereo setup. We refer to these as **Stereo A** (origin at Camera 1) and **Stereo B** (origin at Camera 3). Each stereo pair defines its own coordinate system based on the camera's angle and position, illustrated in Figure 35.

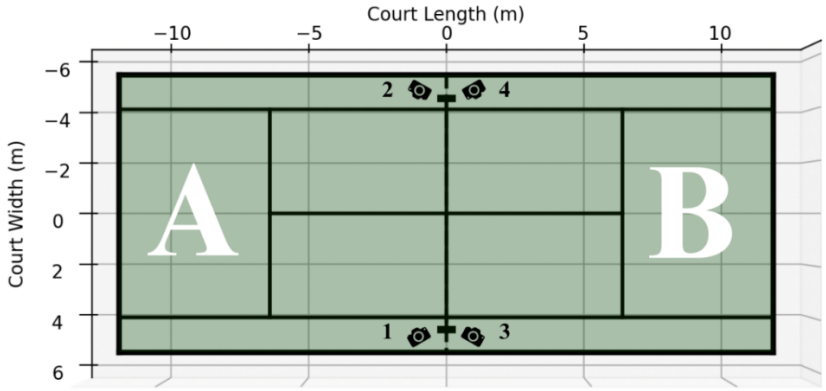



*Figure 35: Stereo cameras configuration*

Because the output coordinates are given in pixel units, we must convert them to a common measurement for further processing, in this case metres. We also observed significant fluctuations in the ball's depth throughout its flight, an issue clearly visible when we plot the data in a 2D Space (see Figure 36) and heavily studied and addressed in previous research [49].

## 6.2 Preprocessing and Normalisation of Stereo Output (LE)

### 6.2.1 Savitzky-Golay Filter and Polynomial Fitting of Depth (LE)

In preparation for modelling a full shot, we first segmented a partial shot captured by Stereo A. As shown in Figure 36, the raw depth values did not exhibit the smooth, continuous trajectory we would expect for a ball in flight. To address this, we initially applied a Savitzky-Golay filter, an algorithm that reduces noise by fitting successive subsets of adjacent data points with a low-degree polynomial [50].

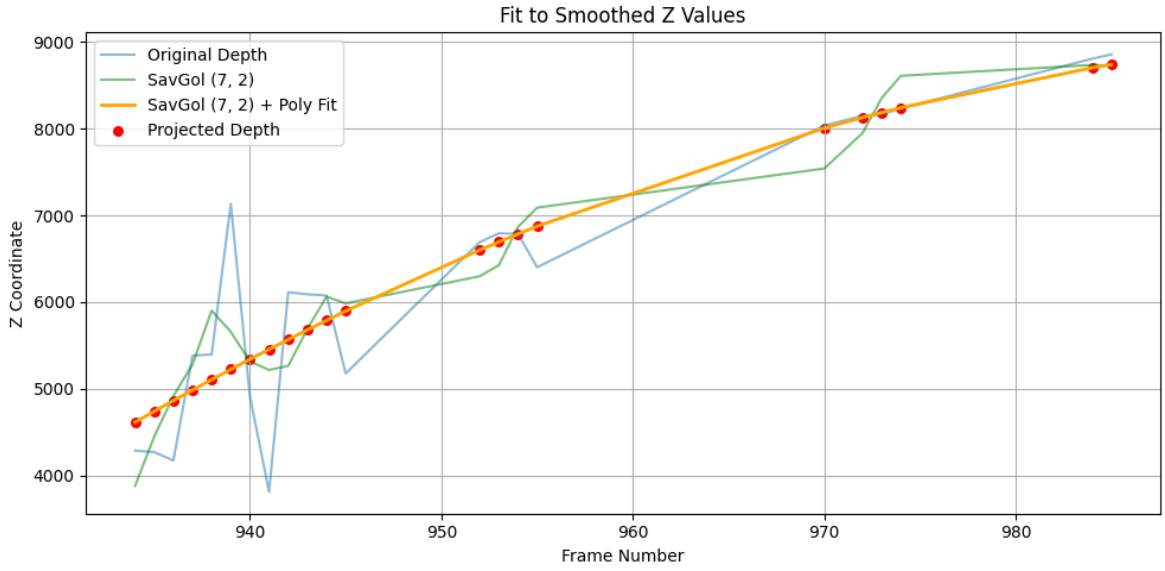



*Figure 36: Depth smoothing using Savitzky-Golay filter and polynomial fitting*

However, the filtered data still did not provide the continuity in depth we were seeking. We thus went further by fitting a second-degree polynomial to our data points and re-evaluated each point at its original frame index, as illustrated by the red markers in Figure 36. This approach yielded a more coherent depth profile, and when evaluated against our ground truth, the depth error decreased by 76% for this shot.

### 6.2.2 Polynomial Fitting of X vs. Z Depth (LE)

The next step we took was ensuring that the $x$ values of the points remained continuous as the ball moved along the $z$-axis (depth). We deferred the $y$ coordinate at this stage because it was often discontinuous due to events like bounces. We considered two main approaches:

1. **Removing Anomalies via Gradient Threshold** (Approach 1)
   We identified frames as anomalous if the gradient $\frac{\Delta x}{\Delta z}$ deviated too sharply between consecutive points. This was done by applying a threshold $\tau$ for the allowable gradient change:

$$\left|\frac{\Delta x_{i+1}}{\Delta z_{i+1}} - \frac{\Delta x_i}{\Delta z_i}\right| > \tau \Longrightarrow anomalous\ frame \tag{13}$$

2. **Fitting $x$ to $z$** (Approach 2)
   Instead of discarding outliers, we modelled $x$ as a polynomial function of $z$, which we already established as continuous over time. This made the $x$-coordinate smoother while retaining the overall trajectory trend.

After applying both methods, we observed a significant drop in accuracy with the anomaly-removal approach (Approach 1), especially for faster shots with only ~8 frames from net to player. Conversely, slower shots contained a higher proportion of anomalous frames, yet they were still easier to correct using the underlying polynomial trend. As a result, we found Approach 2 to be a more stable and robust solution.

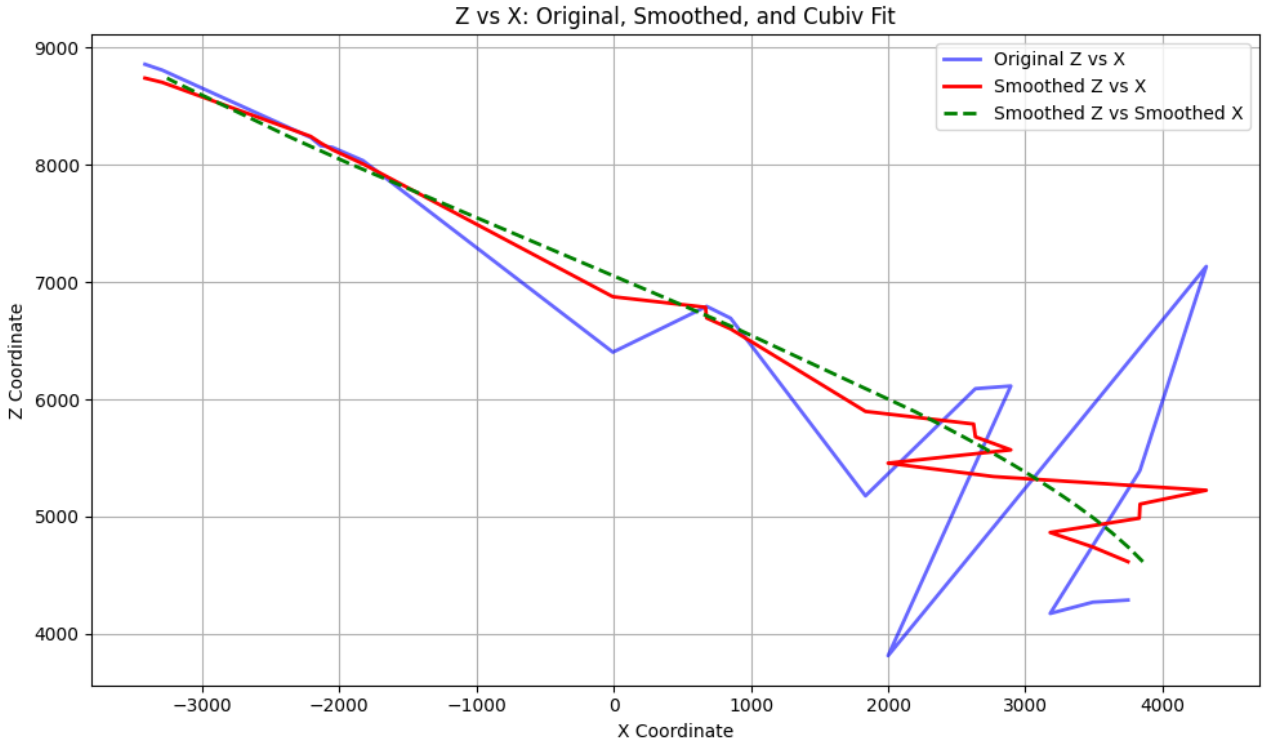



*Figure 37: z vs x coordinates smoothing comparison across original and polynomial-corrected values*

The outcome of Approach 2 is shown in Figure 37, where the red line depicts the unfiltered $x$ values against the smoothed $z$ values, and the green line shows the polynomial-smoothed trajectory. This smoothing is especially noticeable when looking at the shot from a top-down perspective. An additional view of these points, prior to any coordinate translation, is presented in Figure 38.

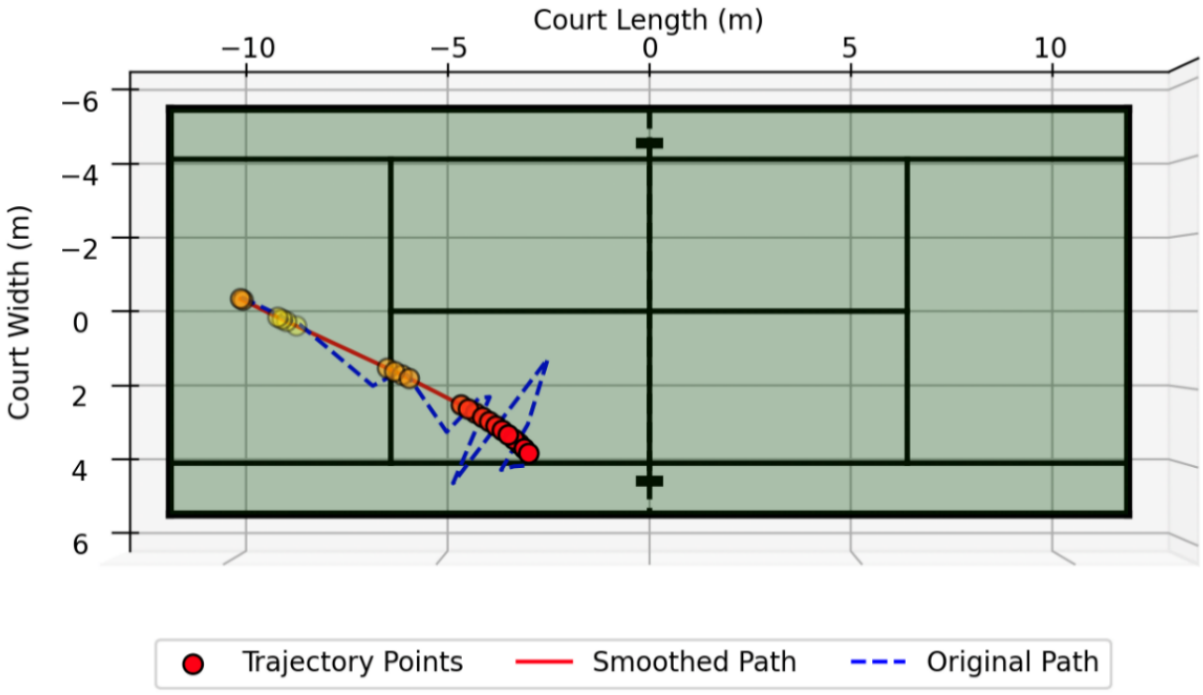



*Figure 38: Top-down trajectory comparison of original and smoothed ball path*

### 6.3 Transformation to Unified Global Coordinate System (LE)

#### 6.3.1 Rigid Transformation (LE)

After processing a single shot, we moved onto transforming the outputs from both stereos into a unified coordinate frame. This process involved applying rotation matrices to account for each camera's mounting angle, translating the coordinate frames to a shared origin, and converting the raw pixel units into metres. The mathematics for these steps is summarised below.

1. **Rotation about $x$ and $y$-axis**
   We begin by rotating a vector $rA$ (from stereo A) by $\theta A$ around the $y$-axis:

$$rA^{(y)} = R_y(\theta A)rA \tag{14}$$

   where

$$R_y(\theta) = \begin{pmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{pmatrix} \tag{15}$$

   We then rotate the resulting vector $rA^{(y)}$ by $\phi A$ about the $x$-axis:

$$rA^{(y,x)} = R_x(\phi A)rA^{(y)} \tag{16}$$

   where

$$R_x(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\phi & -sin\phi \\ 0 & sin\phi & cos\theta \end{pmatrix} \tag{17}$$

   A similar procedure applies for any vector $rB$ from Stereo B:

$$rB^{(y)} = R_y(\theta B)rB,\ rB^{(y,x)} = R_x(\phi B)(rB^{(y)}) \tag{18}$$

   By chaining these two rotation operations, we can reorient each stereo pair's local coordinates to align with a desired global reference frame.

2. **Conversion to Metres**
   After the rotations, we convert pixel units to metres. Empirically, each coordinate dimension is divided by 1000, except that the $y$-coordinate is negated to align the axes properly.

$$rA^{(scaled)} = ScaleA * rA^{(y,x)} \tag{19}$$

3. **Translation to a Common Origin**
   Finally, we translate each coordinate set, so they share an origin. In code, these offsets are:

$$offsetA = \begin{pmatrix} -0.15 \\ 1.4 \\ -6.7 \end{pmatrix}, \qquad offsetB = \begin{pmatrix} 0.15 \\ 1.4 \\ -6.7 \end{pmatrix}$$

   Hence, the final global-coordinate points become:

$$rA^{(final)} = rA^{(scaled)} + offsetA,\ rB^{(final)} = rB^{(scaled)} + offsetB \tag{20}$$

The results of these transformations are illustrated in Figure 39.

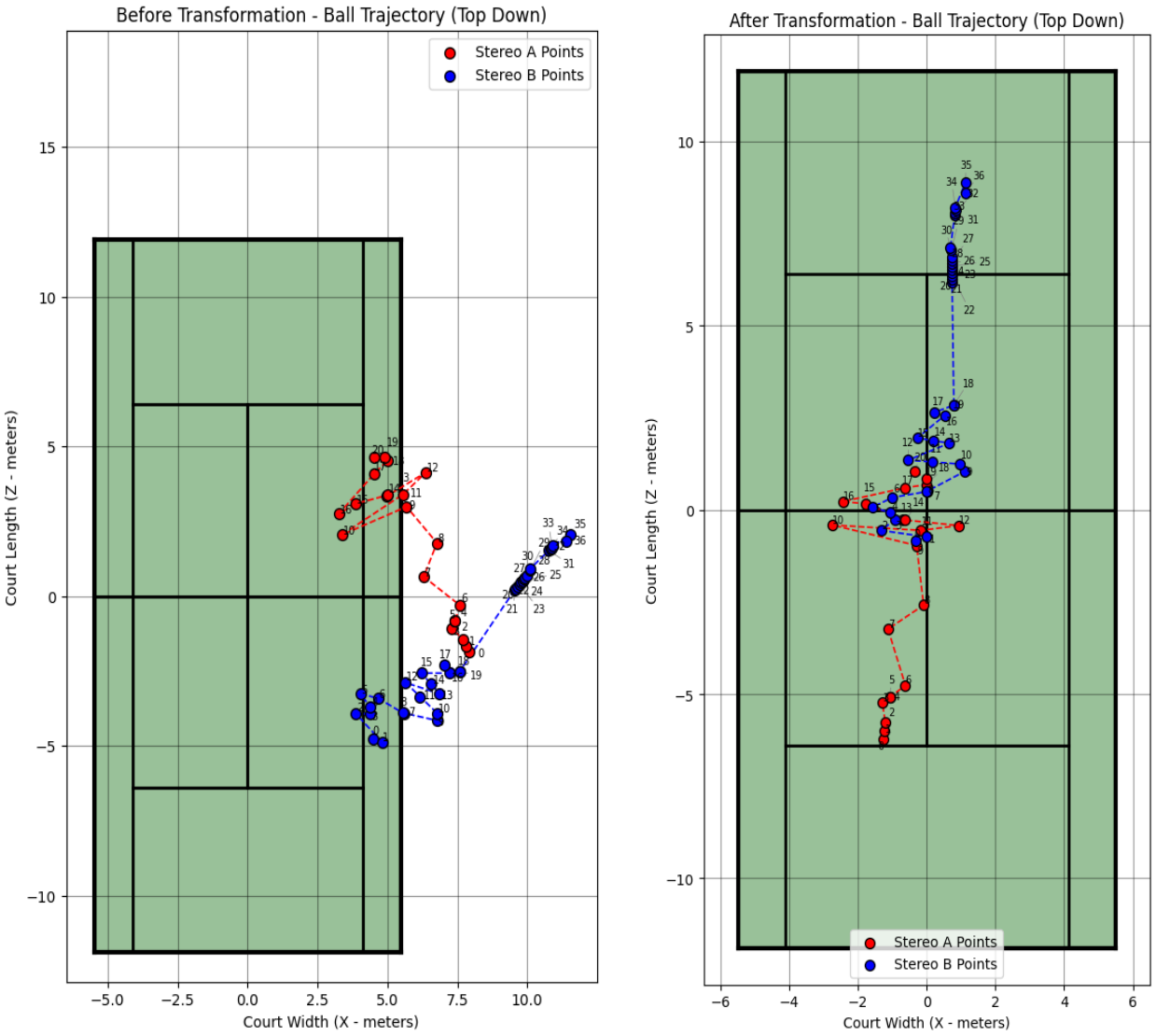



*Figure 39: Ball trajectory before and after rigid transformation to global coordinates*

### 6.3.2 Refinement and Noise Reduction Techniques (LE)

After transforming the coordinates into a shared global coordinate system, we applied a similar methodology to that described in Section 6.2 to reduce the errors introduced by the stereo cameras. The results for the shot outlined in Figure 39 are shown in Figure 40.

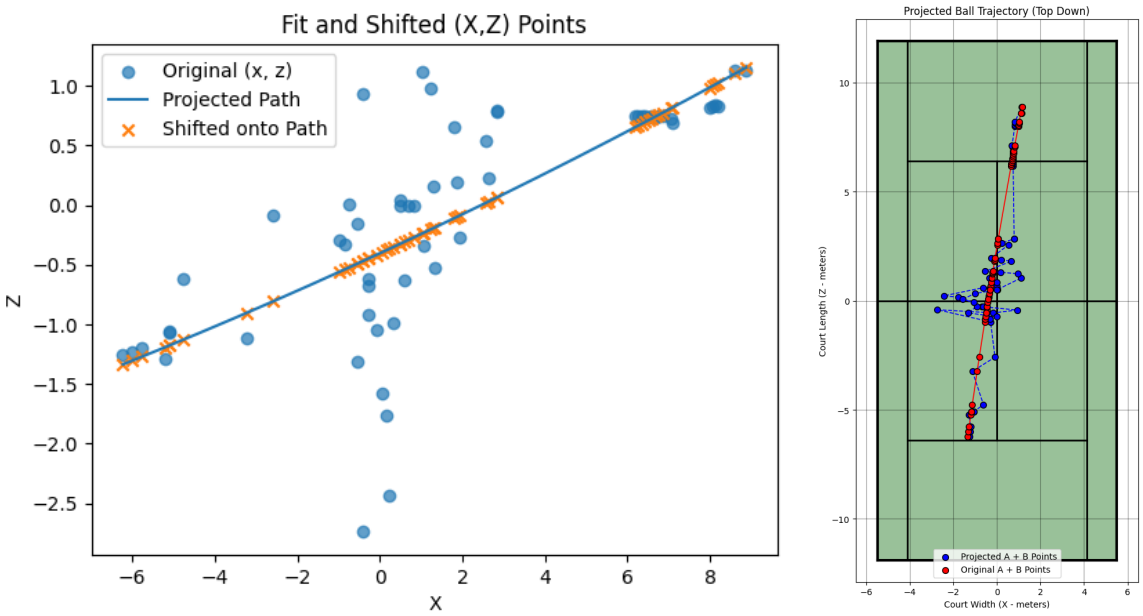



*Figure 40: Refinement of projected points using noise reduction and path alignment*

## 6.4 Automation of Shot Detection (LE)

The process described in previous sections was carefully implemented and tested on a collection of pre-selected shots. To scale this approach and enable retroactive automation, we needed to identify the start and end points of each shot based on the ball's 3D position across consecutive frames. This was achieved by analysing changes in gradient and applying thresholds to detect significant variations indicative of a shot beginning or ending.

We applied this method to a curated dataset of 40,000 frames from over two hours of collected match footage. Figure 41 shows the algorithm used to segment continuous play into a discrete dataset of shots.

This segmentation formed the basis for the simulation and machine learning steps described in Sections 6.5 and 7.

**Algorithm 1:** Shot Segmentation from Stereo Camera Detections

**Input:** StereoA, StereoB: detections with `(frame_no, position, confidence)`

**Output:** List of segmented shots

Merge and Sort StereoA and StereoB by frame, keeping the detection with higher confidence;

Initialize empty list `shots`, `current_shot`;

Set `prev_dir` to None;

**for** *each pair of consecutive positions* $(p_1, p_2)$ **do**

- Compute direction $\vec{d} \leftarrow \text{Normalize}(p_2 - p_1)$;
- **if** `current_shot` *is empty* **then**
  - Append $p_1$ to `current_shot`;
- **if** `prev_dir` *is None* **then**
  - Set `prev_dir` $\leftarrow \vec{d}$;
- Compute angle change $\theta \leftarrow \text{AngleBetween}(\vec{d}, \texttt{prev\_dir})$;
- **if** $\theta$ *is small* **then**
  - Append $p_2$ to `current_shot`;
- **else**
  - **if** `current_shot` *is long enough* **then**
    - Add `current_shot` to `shots`;
  - Start new `current_shot` with $p_2$;
  - Reset `prev_dir` $\leftarrow$ None;
- Update `prev_dir` $\leftarrow \vec{d}$;

**return** `shots`;

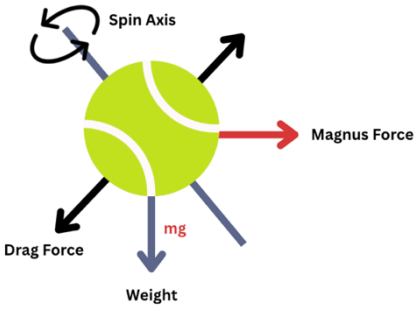
*Figure 41: Shot segmentation algorithm based on stereo detections*

## 6.5 Simulation and Up-Sampling of Shots (LE)

After automating the shot detection process, we obtained a series of refined and segmented shots. Each shot was labelled with its corresponding play outcome. To accurately predict the outcome of a play and validate our methodology, the next step was to model the trajectory and determine its precise landing point. This required the use of various machine learning techniques, as detailed in *Section 7*. However, before applying these methods, we first needed to increase the volume of available data (i.e., trajectories) through up-sampling as follows.

### 6.5.1 Modelling Ball Forces (JF + LE)

We began by researching the forces acting on a tennis ball [51] [52]. A free body diagram showing these forces is presented in Figure 42.




$$\vec{F}_{drag} = \frac{1}{2} C_d \cdot \rho \cdot A \cdot v^2 \cdot \hat{v} \tag{21}$$

$$\vec{F}_{magnus} = \frac{1}{2} \cdot \rho \cdot A \cdot v^2 \cdot C_L \cdot \frac{\vec{w} \times \vec{v}}{|\vec{v}|} \tag{22}$$

*Figure 42: Free body diagram and governing equations of ball forces*

At any given moment, gravity creates a constant downward acceleration of 9.8 $m/s^2$. Air resistance, or drag, acts in the direction opposite to the ball's velocity. It is influenced by the air density, the drag coefficient, and the ball's velocity/cross-sectional area. The Magnus force arises from the interaction between the ball's spin and the surrounding air. It is proportional to the cross product of the ball's angular velocity and the fluid velocity, producing a lift force perpendicular to both the spin axis and the direction of motion. Both the drag and magnus effects are non-linear and are modelled using the equations shown in Figure 42.

### 6.5.2 Simulation Data (JF + LE)

Building upon the research and testing in *Section 6.5.1*, we developed a program to simulate shots by randomly sampling initial conditions, including starting position, spin, ball velocity, and sampling frequency. The program then iteratively constructs the trajectory using Euler's method [53] (see below) for projectile motion constrained by the outlined physical parameters. Here, acceleration $\vec{a}_t$ is calculated from total forces, cascading updates through both velocity $\vec{v}_{t+1}$ and position $\vec{x}_{t+1}$ at the next time step relative to the current.

$$\vec{a}_t = \vec{a}_{gravity} + \vec{a}_{drag} + \vec{a}_{magnus} \tag{23}$$

$$\vec{v}_{t+1} = \vec{v}_t + \vec{a}_t \cdot \Delta t \tag{24}$$

$$\vec{x}_{t+1} = \vec{x}_t + \vec{v}_{t+1} \cdot \Delta t \tag{25}$$

Collating both *Section 6.5.1 and 6.5.2* into a single python module allowed us to begin building trajectory prediction models based solely on the simulated data, with Figure 43 showing examples for a simulated backspin backhand cross-court, and a topspin forehand.

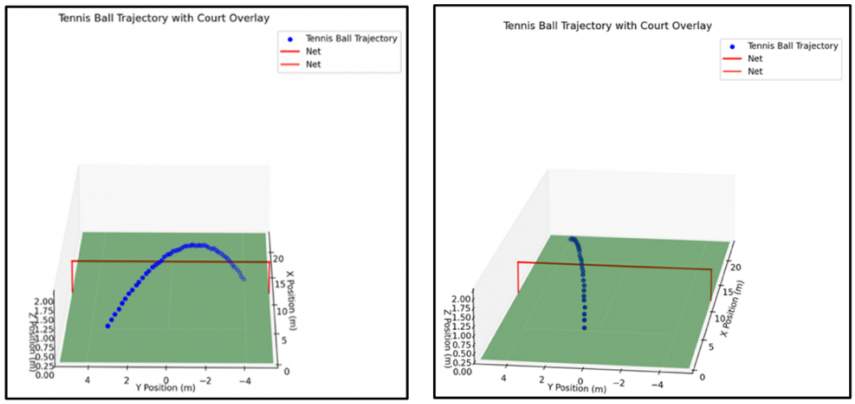



*Figure 43: Simulated backhand slice and forehand top spin*

The main limitation of this approach is its mathematical perfection, exactly fitted to physical models and unrealistic for real tennis systems. This would be addressed by substituting the simulations with further data collection or training the model on publicly available footage. However, for this project, we were limited by our access to the courts, needed to recalibrate the triangulation pipeline described in *Section 5*. As such, we needed a way to both artificially produce large amounts of training data while maintaining real world restraints and randomness, described in the following section.

# 7 Machine Learning Trajectory Prediction

## 7.1 Trajectory Generation (JF)

We used the function already created during physics-based modelling to construct test trajectories with randomised initial spin, position, and velocity parameters. A training dataset of 20,000 trajectories, each bound to land inside a 1-meter boundary surrounding the entire court was established. The simulated synthetic input data were assumed to be captured at 60 fps with a random position error of approximately 7cm, due to quantisation and motion blur. This uncertainty was added as random noise to each point in the 3D trajectory. The development, training, and evaluation of neural network models now follows. A sample visualisation is plotted on a 2D (*x, z*) graph depicting the random noise points surrounding a true trajectory as shown in the following Figure 44.

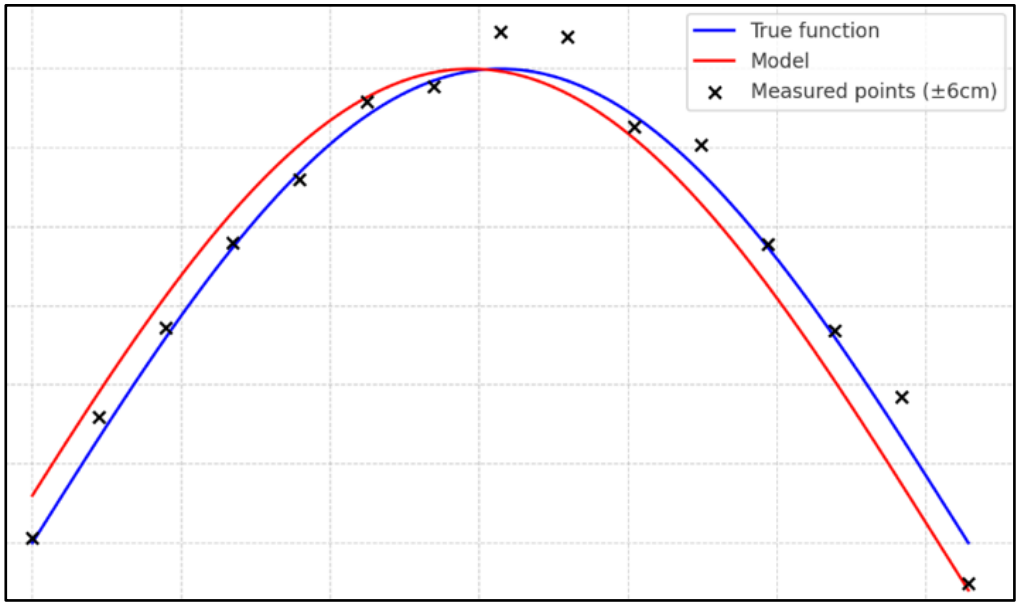



*Figure 44: Erroneous example*

## 7.2 Initial Sliding Window Approach (JF)

Our initial approach was to develop a model able to forecast the remainder of a trajectory given any random initial starting window [54] [55], until the z-value was less than 0 (ball reached ground level). The below Figure 45 highlights an example training scenario, of which we generate 10,000 random samples. During every iteration, an acceptable starting point is randomly selected, with corresponding z-value above ground and truncated until it reaches the ground. After the dataset generation, the input set (X) is populated with the initial 4 to 8 points of each trajectory (based on the window size selected), and the target set (Y) consists of the whole trajectory the model tries predicting.

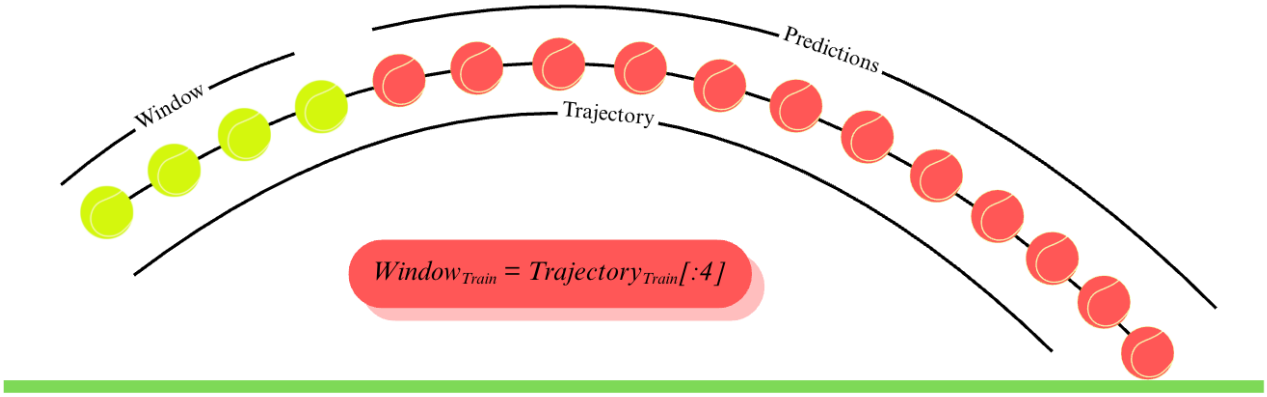



*Figure 45: Building training windows*

A Seq2Seq encoder/decoder [56] structure was also needed to handle inputs of differing sizes to the outputs. This approach, albeit promising ended up struggling as an active window of sizes 4-8 couldn't detect higher order effects like spin accurately, resulting in an extremely high error predicting the landing point.

### 7.3 Prototyped GRU with Frequency Scaling (JF)

We're now introducing the concept of frequency scaling [57]. The idea is to model the ability to reconstruct or predict a higher frequency sequence from one sampled at a lower rate. To do this, we take one for every `FREQUENCY_SCALER` point from the high frequency trajectory (240 fps), effectively down sampling it. To simulate quantisation and motion blur, we then add a random error of 6-7 cm to each down sampled point. This trajectory simulates the expected output from the object detection, calibration and triangulation pipeline, an erroneous 60 fps trajectory shown by Simulating Data in Figure 2. Next, we interpolate this noisy, lower-frequency sequence back to the original length, resulting in two sequences of equal size: one clean and one noisy. The goal is to train a model that can take a distorted 60 fps sequence as input, interpolate and learn to predict the underlying, undistorted trend that passes through those points at a higher frequency. As shown in Figure 46.

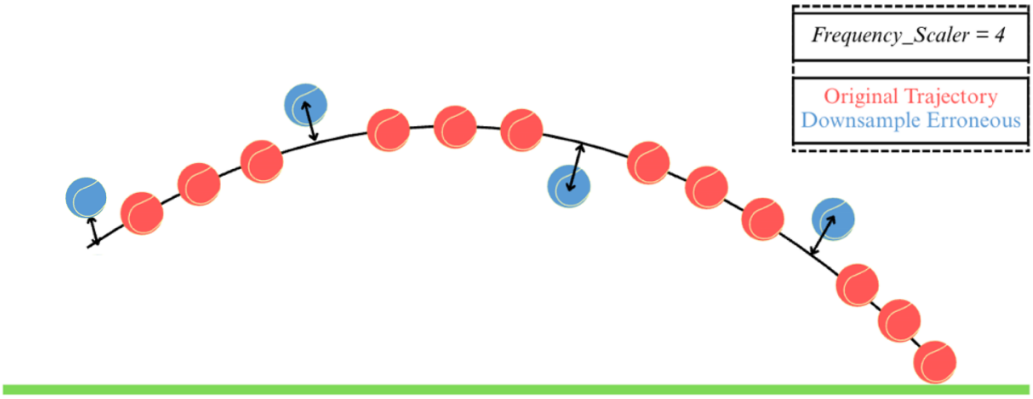



*Figure 46: Erroneous trajectory simulation using frequency scaling and noise*

Below shows the simple model diagram for an initial GRU [58] attempt as well as an example prediction. This model has many major benefits mainly it's training efficiency and few parameters making it a useful prototyping baseline. Carrying out GRU model training, taking the erroneous sequence as input and comparing to ground truth, the predicted results are still highly erroneous, incorrectly predicting the undistorted trend. This clearly needed significant improvements. We introduced two primary metrics to evaluate the progression of my models: Endpoint Error [59] and DTW (Dynamic Time Warping) error [60]. The Endpoint Error measures the average distance between the predicted and actual landing points. Meanwhile, DTW error quantifies how closely the overall shape of the predicted trajectory matches the ground truth, depicting temporal alignment. The GRU approach scores an average Endpoint Error of 68.6cm (1.d.p) and a DTW error of 106.9 (1.d.p) both notably poor. The Endpoint Error specifically we want to end within approximately one ball diameter (6-7cm), with a sample trajectory shown in Figure 47.

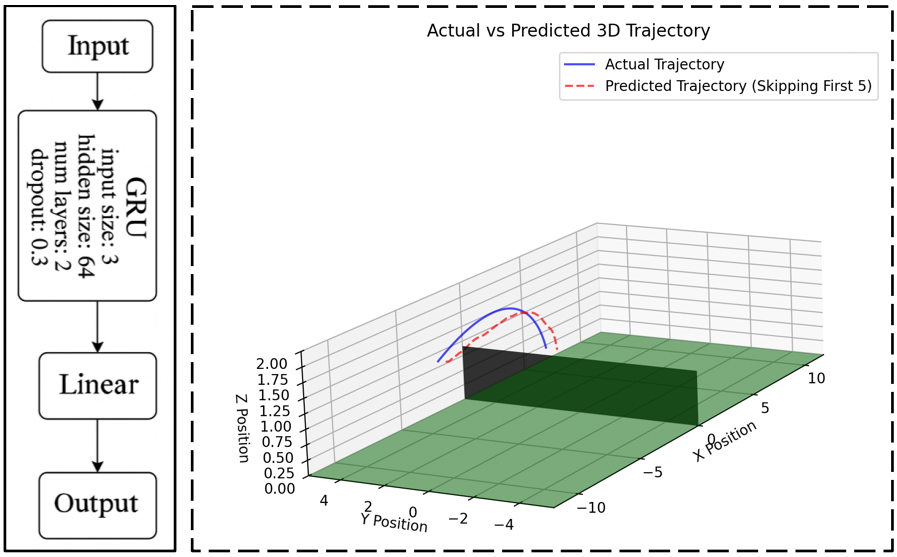



*Figure 47: Initial GRU prediction example*

### 7.4 Landing Point Tailored Model Development (JF)

To compare final models, we introduced a saving mechanism allowing separate modules to load from a model folder and compare their final metrics. The model options considered were LSTMs, Temporal Convolutional Networks and Bi-Directional LSTMs (As shown in Figure 48).

```
if os.path.exists(MODEL_PATH) and not RETRAIN:
    print("Loading pre-trained model")
    model.load_state_dict(torch.load(MODEL_PATH, map_location=device))
else:
    dataset = TrajectoryDataset()
    dataloader = DataLoader(dataset, batch_size=1, shuffle=True)
    train_model(model, dataloader)
    torch.save(model.state_dict(), MODEL_PATH)
    print("Model saved successfully")
```

*Figure 48: Model saving mechanism*

An LSTM [61] is the standard unidirectional recurrent model, that processes sequences from start to end. It is good for capturing long-term dependencies and for sequential predictions that evolve overtime. Figure 49 includes an LSTM unit, which takes the current input, the previous hidden state, and a memory value from the past. The forget gate decides what old information to discard, while the input gate updates the memory with new information, some of which filtered into new hidden state. LSTMs cannot leverage any future context and are prone to overfitting, speculated to be purely a good starting point for the frequency scaling problem.

Unlike RNNs, Temporal Convolutional Networks (TCNs) [62], do not process sequentially. Instead, they capture temporal dependencies using 1D convolutions across time. TCNs leverage dilated convolutions to exponentially expand their field of reception. For example, with a dilation rate of 2 (used in our case) the model skips every other time step, enabling access to long-range temporal patterns without requiring a deep architecture. This uses a sequence of convolutional blocks, alternating ReLU and Convolution operations, each with scaling dilation lookback.

A Bi-LSTM [63] is a variant of the normal unidirectional LSTM that reads sequences in both directions simultaneously. This allows the model to learn from both past and future time steps, enabling better learning of sequence dependencies. A Bi-LSTM cell contains two different hidden states for each direction, which are concatenated to produce the ultimate output. Bi-LSTMs can learn denser contextual representations, performing better on tasks where future context is important. However, they require additional computational resources and complexity. The functional blocks for our TCN and Bi-LSTMs are highlighted below [64].

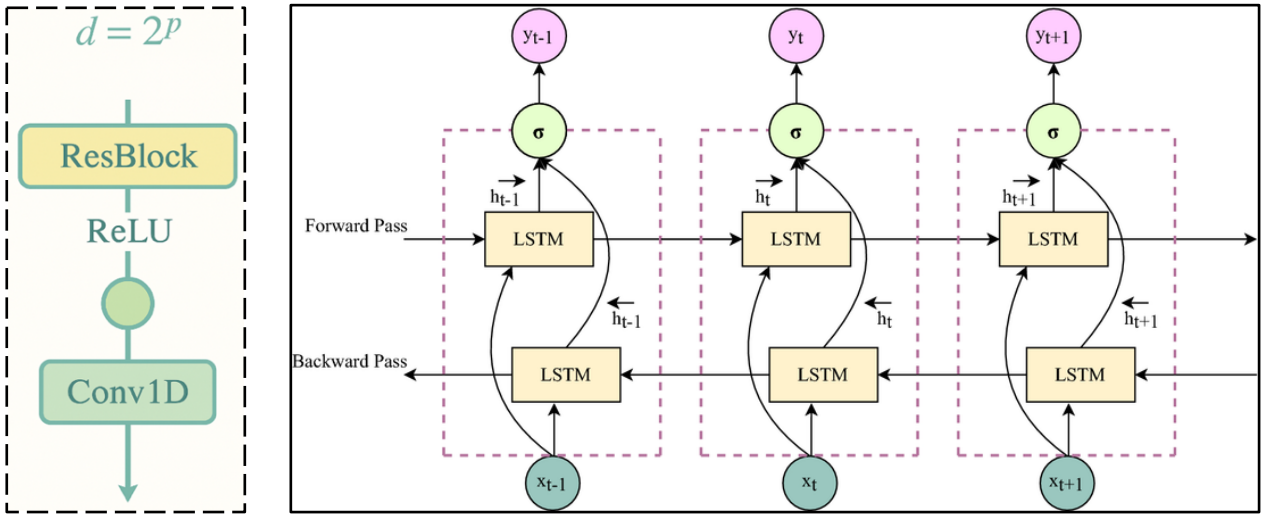



*Figure 49: TCN and Bi-LSTM Single Unit* [64]

We trained these models using an Adam optimiser and an MSE loss criterion, supplemented by custom loss functions for endpoint and smoothness, tailored to our specific problem. Hypotheses were formed expecting the Bi-LSTM to perform the best, as its bidirectionality would significantly help spot the underlying trend.

### 7.5 Final Prediction Metrics and Model Selection (JF)

Figure 50 shows graphical metric conclusions across the prior discussed model options.

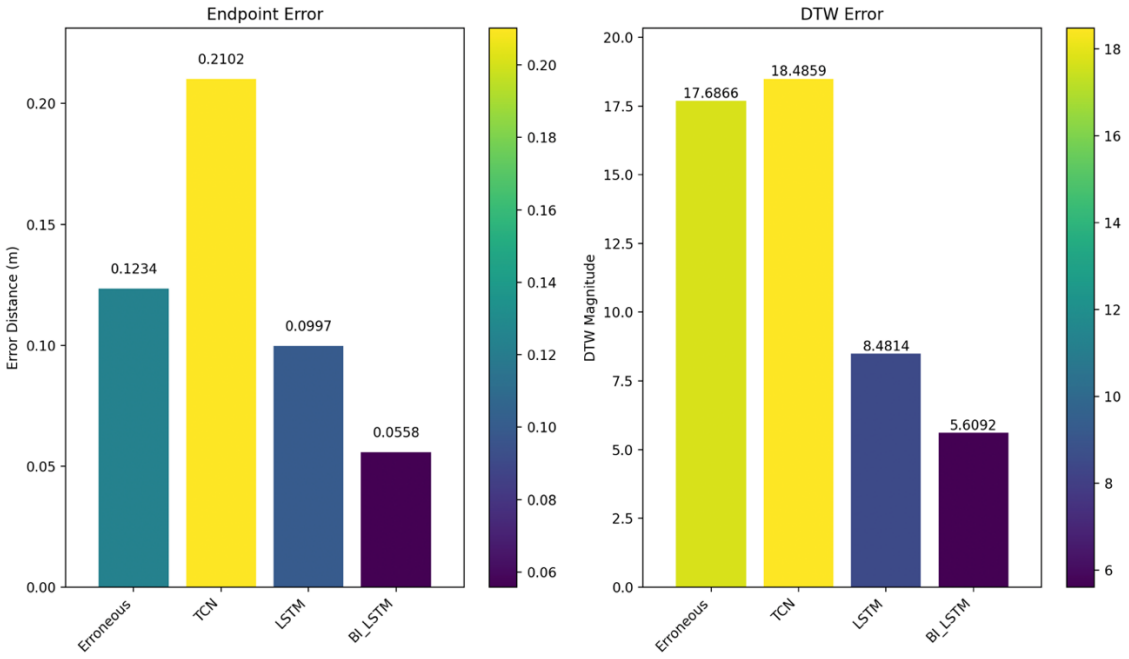



*Figure 50: Final endpoint and DTW errors*

The endpoint error, as predicted, was lowest for the Bi-Directional LSTM, halving the disparity shown in the training erroneous sequence and within the acceptable range we desired. Our final landing point prediction accuracy came to within 5.6 cm (1.d.p). The dynamic time warping also showed the greatest improvements after passing through the Bi-LSTM model, showing marked smoothening and a statistically insignificant final DTW of 5.6 (1.d.p). Notably the endpoint error halving in comparison to the original erroneous 60 fps trajectory showed the clear benefit of this frequency scaling approach and confirmed our model should be a Bi-LSTM, whose 3D prediction output is shown below.

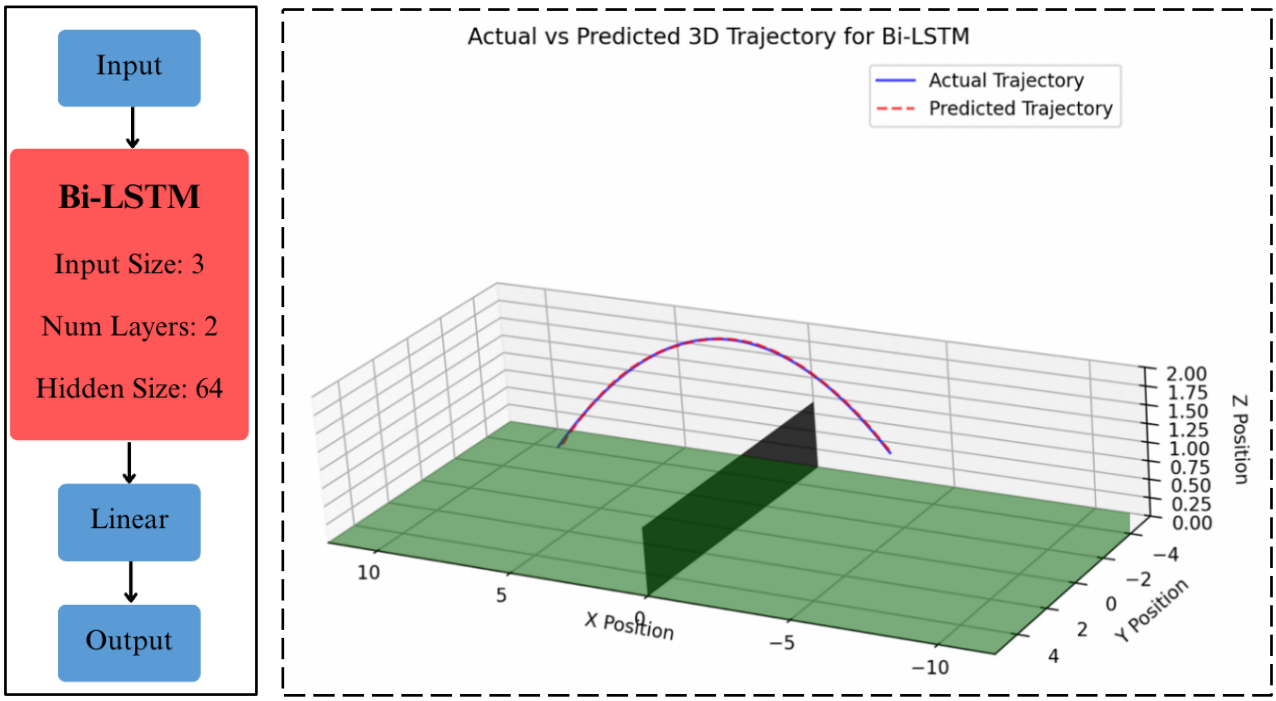



*Figure 51: Final model and prediction depiction*

Two additional metrics were also explored for comparing the three models, our custom confidence score and mAP50/mAP95 ideas. The calculation for confidence score is shown below, equal to the inverse of average lateral deviation.

$$confidence = \frac{1}{1+distance_{avg}} \tag{26}$$

We also engineered metrics like the mean average precision (mAP50 and mAP95 [65]) for object detection. These are custom binary accuracy metrics that assess how close a predicted endpoint is to the ground truth. The mAP50 evaluates predictions within a 12cm threshold, while mAP95 uses a stricter 6cm threshold, indicating higher precision. Again, our Bi-LSTM, comes out best having an average confidence of 93.76% for every point predicted, with 90% of endpoints points within 12cm and 40% within 6cm.

| Model Type | Conf | mAP@50 | mAP@95 |
|---|---|---|---|
| TCN | 0.8546 | 0.5000 | 0.1000 |
| LSTM | 0.9277 | 0.7000 | 0.2000 |
| BI_LSTM | 0.9376 | 0.9000 | 0.4000 |

*Figure 52: Additional metrics*

### 7.6 Real-Time Enhancements and Interactive Noise/Occlusion Simulation (JF)

We developed a script that loads a pretrained Bi-LSTM trajectory prediction model, then applies three optimisation techniques. First, we prune [66] to remove unimportant LSTM weights. We save the pruned model in the ONNX format [67], which runs 2x to 4x faster on a CPU. Then use dynamic quantisation [68], from float32 to int8, which shrinks the model by approximately 75%. All these improvements would significantly help if we were to evolve our system into a real-time equivalent. Finally, we built a Streamlit [69] application designed to artificially introduce greater noise or occlude a significant portion of the data, allowing us to see how well the resulting trajectory came out, shown in Figure 53. As highlighted in the example usage below, increasing the noise to 8cm and predicting with 35% of data missing still yields an accurate prediction and landing point. Potential future expansions of this section include hyperparameter optimisations using software integrations such as Optuna [70] and further model expansion into higher-level fields including Transformer models [71]. Due to the required resources for larger model training and project timeline restraints these were not explored further.

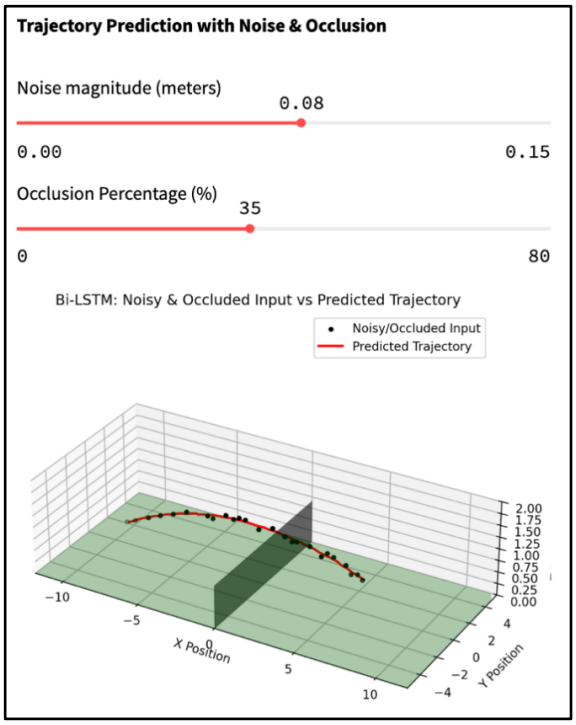



*Figure 53: Streamlit Noise/Occlusion Application*

# 8 Conclusion

We will end by discussing the final achievements at each stage of the pipeline, as well as areas in which our work could be developed upon, before including an additional research task in *Section 9.*

## 8.1 Summary of Achievements (JF)

When considering object detection, we aimed to balance computational efficiency and accuracy while detecting the ball within each frame. We ended up leveraging a sequence of methods including cross validation of YOLO and Background Subtraction with various fall-back methods including Lucas-Kanade optical flow tracking and HSV/Hough circle detection. After many iterations, aiming to maximise successful detection rates and reduce average inference time, our final metrics for this hybrid approach included 100% detection rate and a 3.5ms inference time, both exceptionally optimised for tennis line calling. The substantial improvements across development are highlighted below in Figure 54.

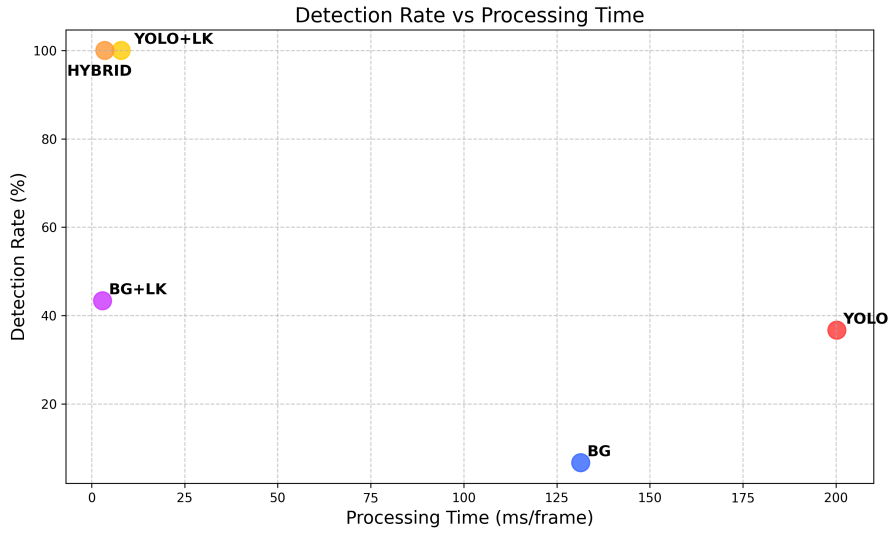



*Figure 54: Object Detection Progress*

Next, discussing calibration outcomes, we consider each camera individually and their combination to form stereo pairs. Intrinsics, mapping each camera's pixels into 3D space, required threshold considerations to optimise the trade-off between the aim of reducing reprojection error while maintaining a high frame count. The final intrinsics decision landed on a threshold of 0.3, with an average reduction in reproduction error of 53.32% across all four cameras while retaining a high frame count. When combining to consider calibrating stereo pairs, the number of initial frames available for extrinsic calibration was limited. Nevertheless, we achieved an average reprojection error of 0.92 (2.d.p) using a threshold of 1, chosen instead of 0.8 to retain as many stereo frames as possible. The improvements in triangulation from using accurate calibration are shown in Figure 55.

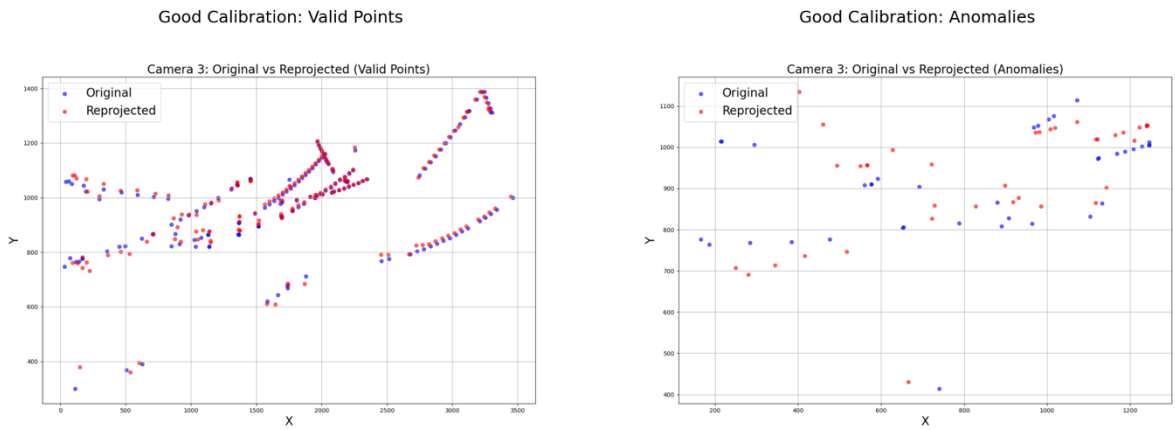



*Figure 55: Anomalies detected from 3D coordinates after Calibration and Triangulation*

Triangulation into the global positioning coordinates resulted in an average of 87% points not being deemed anomalous with an average error of 4.66cm across dimensions. Further work helped with depth smoothening, including Savitzky-Golay Filtering and Polynomial Depth Fitting. The resulting mapping onto a single path for the X and Z dimensions is shown below in Figure 56, culminating our real-world pipeline.

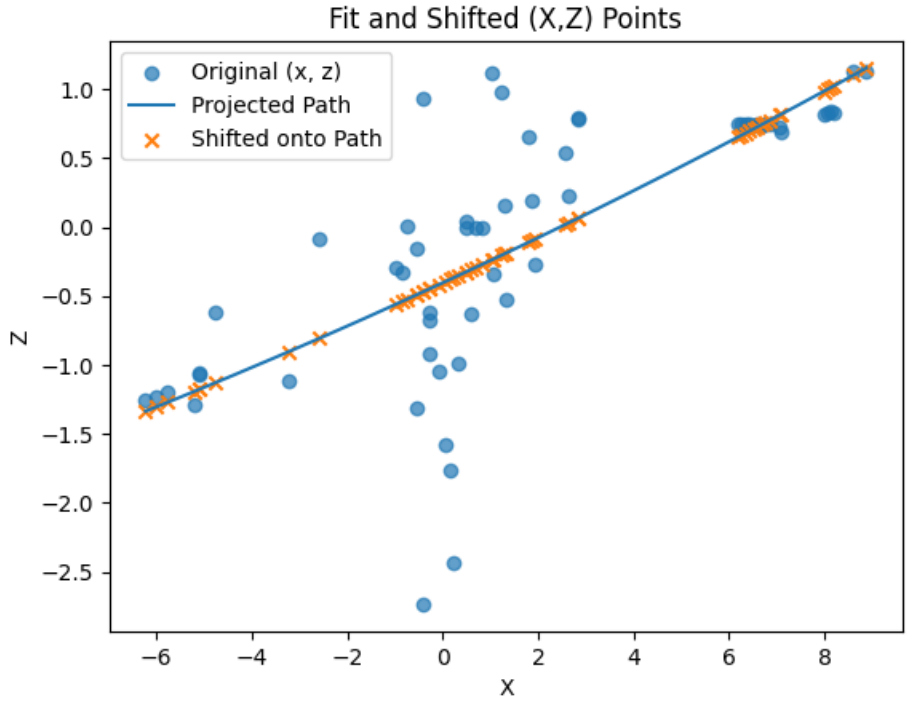



*Figure 56: Final XZ path alignment after triangulation and depth smoothing*

Finally, we must consider the landing point prediction metrics after smoothening and upscaling simulation data. We evolved our procedure through both sliding windows and many neural networks, finishing with the ability to predict a smooth, upscaled, 240 fps trajectory given an erroneous 60 fps one. Our final landing point error was within 5.6cm (approximately one ball diameter), with excellent similarity to the original trajectory depicted in Figure 57.

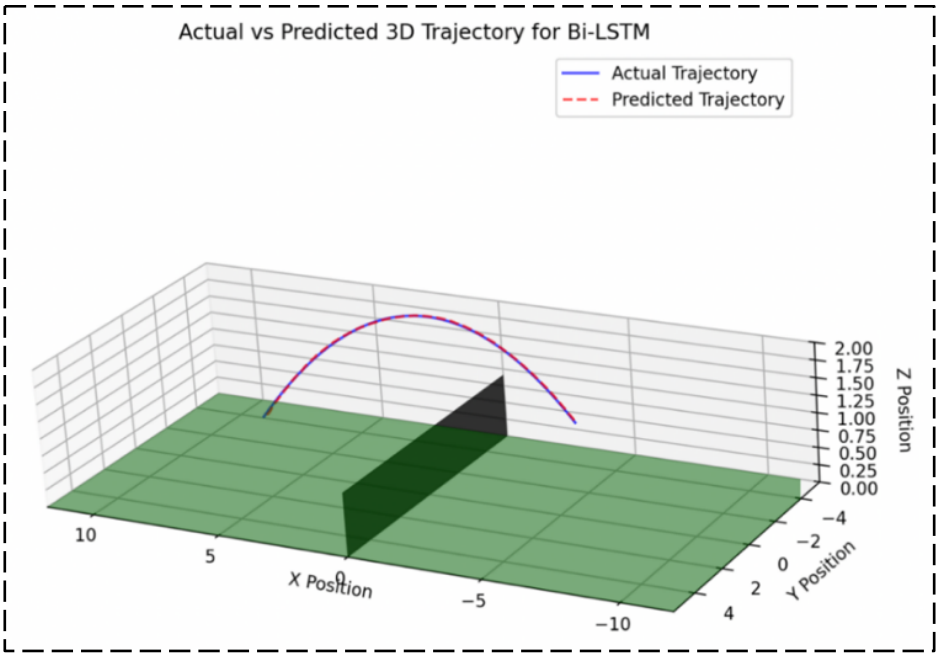



*Figure 57: Predicted vs actual 3D trajectory from Bi-LSTM landing model*

## 8.2 Critical Analysis with Respect to Original Requirements (JF)

This project aimed to develop a low-cost, automated line calling system meeting competitive accuracy standards, using only consumer-grade devices. We met the original objectives *1-5* to an excellent standard, effectively progressing through our pipeline from sensor selection all the way to ball flight prediction. Minimal work has been done to prototype a line calling and scoring system, this being a trivial task after each landing point had been accurately found, in comparison to those prior in *Sections 1-8.* We were unable to explore a real-time user interface due to both time and resource restraints, prioritising the delivery of surrounding work to the highest possible standard. Completions are summarised in Figure 58.

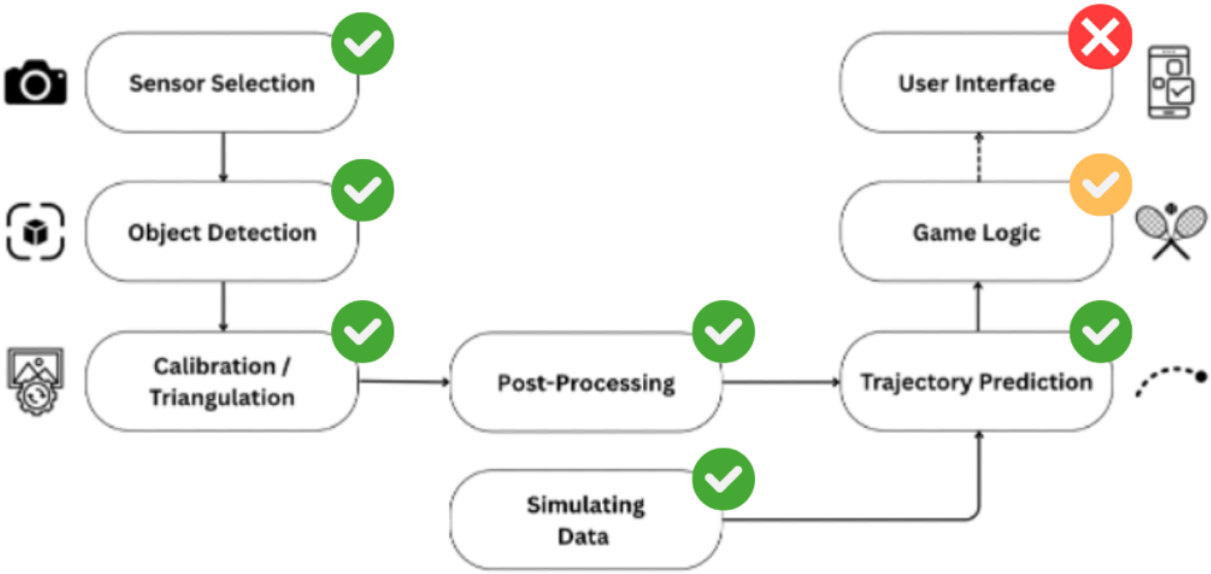



*Figure 58: Final Project Progress*

## 8.3 Potential for Future Research (JF)

To progress our work further, there are many avenues to explore beyond the finalising of a polished user interface. Potential optimisations are included in Figure 59.

| Improvement Area | Proposed Method | Expected Impact | Priority |
|---|---|---|---|
| Calibration Automation | Auto calibration for camera intrinsics and inter device extrinsics | Simplified setup and reduced manual error | Medium Term |
| Advanced Temporal Smoothening | Transformer based trajectory prediction | Improved processing capacity and tailoring for high speed motion | Short Term |
| Multi-Camera Fusion | Additional low cost cameras | Enhance accuracy without dramatic hardware cost | Long Term |
| Adaptive Thresholds | Machine learning object detection and reprojection thresholds | Dynamic performance improvements | Medium Term |
| Edge Device Integration | Research acceleration and devices (Raspberry Pi, Jetson Nano) | Improvements tailored to real world UI production | Long Term |

*Figure 59: Potential future improvements with expected impact and priority*

To conclude, we have developed a cost-effective system to predict the landing point of a tennis ball, using consumer grade cameras, built with exceptional performance as the bare minimum at every stage of our pipeline and delivering our original objectives.

# 9 Additional Research

Based on the current program performance analysis method. The drawback is that it is difficult to calculate the exact error rate due to the analysis method still having a level of uncertainty caused by manually measured camera angle and position data compared to actual data. Thus, the analysis result could only provide a general idea of program performance. A more accurate, systematic performance evaluation method should be considered.

## 9.1 Alternative Automation Ground Truth Detection (EL)

Since all flying tennis ball analysis methods cannot avoid error rate, therefore, focusing on the ground truth of the tennis ball landing point is the best strategy, as it only focuses on the properties of the court itself, and any human error will not interfere with this analysis method.

### 9.1.1 Method Selection (EL)

1) Sensor selection and position

For an alternative method within budget, there are a few ways shown below that have been investigated:

One potential approach involves utilising a single wide-angle high-frame-rate camera positioned at a height of 2 meters on one side of the court, shown in Figure 60. Left image. A minimum resolution of 2K is required to ensure performance will be estimated at 2 K, which is out of budget.

However, an alternative method involves using two standard-angle cameras to cover the entire court with a combined image method. But this solution introduces alignment challenges, as precise placement and parallel orientation are required. Failure to achieve will introduce a huge error rate thus not recommended.

Based on the above method, another approach is to place them on each player's corner side and process them separately. Although the solution seems viable, it presents certain challenges. For example, how to correctly identify four reference points on a tennis court for homograph matrix computation [72]. This limitation introduces a potential error rate increase when mapping pixels to real-world coordinates, compared to a single wide-angle camera, which could directly use the tennis court's four corners as the reference points.

But from a cost perspective, standard professional high-frame-rate cameras are available at approximately 300 pounds each, which is within budget, thus, this method is the best approach in the current situation.

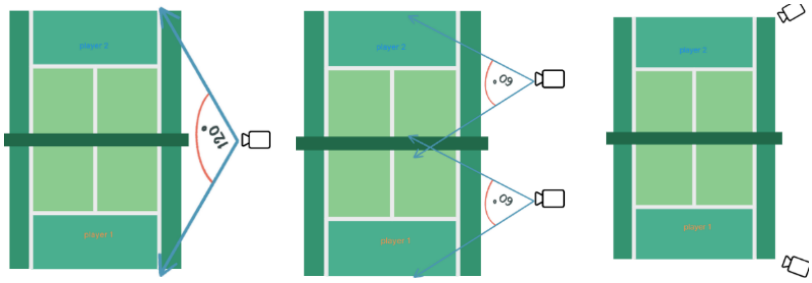



*Figure 60: 3 Method visualisation*

2) Computational Considerations

Based on the method, several computational considerations are shown below that have been considered:

1. Processing 1980x1080 pixels per frame at 240 frames per second presents significant computational demands. To optimise performance, only pixels within the court area need to be analysed, thus eliminating unnecessary computations beyond the court boundaries. To further improve, certain frames can be skipped during processing. If the tennis ball's landing point y increases compared to the former frame, all frames within that sequence should be re-processed to get the exact landing point.

2. Given the distinct colour between the court and the tennis ball, the colour threshold method has been employed to detect tennis ball location instead of a trained AI detection model to reduce runtime.

4. After the tennis ball's landing pixel coordinates are identified, they need to be transformed into real-world coordinates. Traditional tracking methods are impractical in one camera system, thus should ignore the height Z axis value and only focus on the 2D x and y coordinates. Based on the homograph matrix [72], by utilising corresponding points in the image court corners with 3d model court corners to compute a transformation matrix that accurately maps pixel coordinates to 3d model locations.

In this method, four reference points are set similar to Figure 70, due to even distribution of calibration points for homograph mapping. However, in this strategy, finding the exact net corner becomes problematic. A potential solution is focusing on the net itself and using its baseline to establish a linear reference, then following the equation to find the coordinates that sit on the boundary lines. This approach is based on extended projection to estimate the net corners, thus, some degree of error remains inevitable. Nonetheless, this method provides a general and efficient solution across varying court conditions while maintaining a level of accuracy.

### 9.1.2 Method Implementation (EL)

1) Corner Detection

For the line detection method, the court corners can be detected using colour thresholding followed by morphological operations [73] to remove noise. Once the noise is cleaned, the leftmost and bottommost white pixels can be identified as court corners.

As for net corners, based on the net's properties and its position relative to the court, the net baseline middle point can be estimated by identifying the minimum x-axis net colour pixel at approximately 3/4 of the y-axis position. By searching for net-baseline on both sides, a linear data set can be gathered and used to estimate the linear equation. Then, using backwards and forward search until finding the net corners which adjacent to the white boundary lines. Once four reference points are identified, the Homograph Matrix can be computed.

2) Ball Detection

Once the court corners are identified, the possible ball location areas can be restricted to optimise processing and eliminate redundant calculations. Within this region, colour thresholding can be applied to detect yellow pixels, which in the HSV range [25, 80, 80], to [40, 255, 255]. Once the possible ball pixels are identified, two methods have been used to verify the detection:

1. Predefined Bounding Box Method: Since during detection, the first detected tennis ball pixel can only be within (top, left-middle, and bottom), thus detection can be optimised using predefined bounding boxes. The second detection box in Figure 61, middle (Box 2), is the most frequent occurrence. Thus, check box 2 first, then check boxes 1 and 2 for each yellow pixel. The bounding box parameters are determined using sample images, calculating tennis ball total pixels of the closest and farthest landing point situations. Once a valid detected point pixel is confirmed, keep shrinking the bounding box areas until it matches the tennis ball boundaries. Then the centre of the bounding box and the centre bottom pixel can be identified as the tennis ball centre and landing point pixels. The advantage of this method is that the processing speed will be highly efficient in cases with minimal noise. However, in noisy environments, computational overhead increases, and most importantly, if the ball is half visible due to noise, this method is unusable.

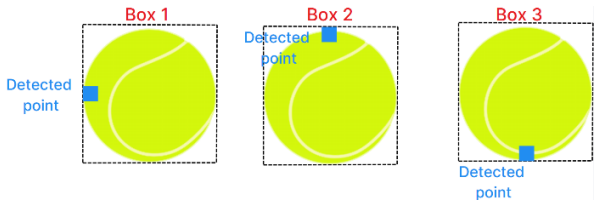



*Figure 61: Predefined bounding box method*

2. Self-defined Bounding Box(Contour) Method [74]: To handle noisy and half visible conditions, this method involves gathering all adjacent pixels that share similar intensity, combine to contours, then eliminate small contours (due to manually computation is not fully optimised increase runtime, alternatively using cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE) which gather all possible Retrieves only external (outermost) contours, Compresses the contour points to top left corner location, width and height of the box instead stores all contour points). After small contours are filtered out, the most similar width and height box is chosen to ensure a circular shape.

This method offers faster processing compared to the first approach. Additionally, cv2.minEnclosingCircle() [75] can be utilised which computes the smallest enclosing circle around a given contour, making it particularly effective when portions of the tennis ball are missing due to non-yellow noise or preprocessing operations, to determine the most suitable centre and radius of the tennis ball, and the radius could be used to estimate the ball's area and validate its shape compared to the contour's area shown in Figure 62. bottom green (if statement) code. The landing point can be determined by adding the radius to the centre coordinate.

```python
# Find contours in the mask, Retrieves only external (outermost) contours,Compresses the contour points instead stores all cont
contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
landingx=0
landingy=0
for contour in contours:
    if cv2.contourArea(contour) > 30:  # Ignore small noises,  ----------------------------------------------------------------

        # Get bounding box
        ballx, bally, w, h = cv2.boundingRect(contour)

        if abs(w - h) < 10:  #-------------------------------------------------------------------------------------------

            # (Optional) Ensure it's circular using min enclosing circle
            center, radius = cv2.minEnclosingCircle(contour)
            #circle_area = np.pi * (radius ** 2)
            #contour_area = cv2.contourArea(contour)
            ballcenter=[center[0],center[1]]

            # if contour_area / circle_area > 0.7:  # A perfect circle would be around 1.0----------------------accuracy if wanted
```

*Figure 62: Self-defined bounding box (contour) method*

3) Transforming Pixel Coordinates to Real-World Coordinates

Once the landing point is identified, the Homograph Matrix is applied to transform pixel coordinates into real-world landing coordinates. To determine the exact landing point, frames can be processed iteratively, and a loop to track the landing point, continuing frame processing until a detected landing point has a higher y-coordinate than the previous frame; the previous frame will be the exact landing point coordinate.

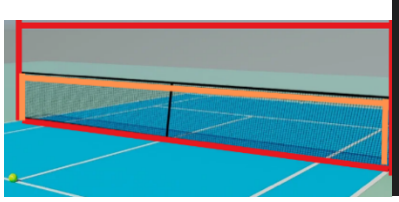
4) Real Landing Point Selection

In this program, the final result should be a series of valid landing points in one round. To ensure the program correctly identifies valid landing points, the following cases have been considered:

1. Outside Boundary: The program must verify whether the landing point lies within the court's four-corner quadrilateral box. Using the straightforward method of limiting the area to a rectangle and two triangles, checking the landing point within the rectangle or two triangles seems time-consuming. Alternatively, the Ray casting method [76] can be used by drawing a virtual ray from the point to check how many points inside the ray intersect the edge of the box; if it is an even number, then inside the box.

2. Rebounce and Round Start: To distinguish between different game rounds and rebounces of the tennis ball after landing, the court is divided into two player sectors, shown in Figure 63. A variable landing_point_detected is introduced. If a new landing point is outside the predefined red box while the previous point was inside, it indicates the ball has hit the current player's court, thus resetting the variable. This mechanism prevents redundant detections until the ball is hit to the current player again. In order to avoid the ball being out of the image, causing an error landing point situation, if the variable is activated meantime can't find the ball in the frame will automatically reset the variable.

3. Net Collision: When the ball is blocked by the net, it may accidentally cross the net boundary and cause incorrect landing_point_detected updates. Expanding the red box may not completely resolve this issue. Instead, define an orange net region box using the net bottom and the top corners, and when a ball landing point is within this area, frames are ignored for a short duration (e.g., 3 seconds or 720 frames). Additionally, to prevent accidental detections from the opposing player holding the ball, previous landing points are also checked to confirm the ball was not on the other side of the court. Although without a second sensor support to prevent accidentally identifying balls landing at the net, a 3-second duration is less than hitting to the opponent side, then hitting back to the player, and finally landing duration, thus it will not affect the landing sequence result. The code implementation is shown in the right image.



```python
if skip == 0 :

    if inside_net_testing(new_landingpoint[0],new_landingpoint[1]) == True and otherside_court_testing(old_landingpoint[0],old_landingpoint[1]) == False : # ------
        skip = 1440 #-------------------------6 second 1440 frame
        print("the ball in net, ignore few frame")

    else :

        if otherside_court_testing(new_landingpoint[0],new_landingpoint[1]) == False and otherside_court_testing(old_landingpoint[0],old_landingpoint[1]) == True :
            landing_point_detected = False

        if landing_point_detected == False and new_landingpoint == [0,0]:
            landing_point_detected = True


        if landing_point_detected == False and old_landingpoint[0] - new_landingpoint[0] > 0:  #--------------- is a landing point
            if inside_box_testing(new_landingpoint[0],new_landingpoint[1]) == True : #------------ this is inside court
                print("real landing pont")
                landing_point_series.append(new_landingpoint)
                landing_point_frame_series.append(series)
                landing_point_detected = True
            else :
                landing_point_detected = True
```

*Figure 63: Opponent and net sectors and real landing point selection implementation.*

### 9.1.3 Improvement (EL)

A) Current Performance Analysis

The current camera settings demonstrate promising results for corner detection, the system successfully identifies the four corners with high accuracy, and for ball detection, it also successfully identifies. But for analysis of the performance of the real-world landing point of the tennis ball, from the farthest to the closest landing point has a 3-cm error rate increase. This error rate is caused by the unpredictable pixel location of the actual landing point within the image. As shown in Figure 64, if the landing point is close to the sensor, the detected pixel calculated red arrow landing point is different from the actual orange arrow pointed actual landing point. Based on this presumption and after testing, if the tennis ball is located far away from the centre and the 4 corners, the error rate will increase to more than 15 cm, which improvement is required.

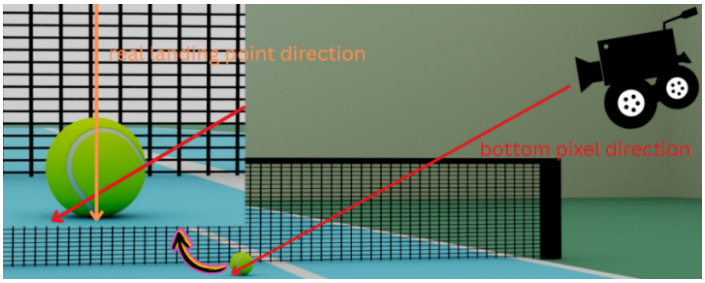



*Figure 64: Real and estimated landing point direction*

B) Improvement

1) To improve colour detection accuracy, the HSV colour space is used instead of RGB. HSV enables better separation of colour from intensity and saturation, allowing the filtering of brightness variations while focusing on the colour itself, thus helping isolate specific colours and ignoring variations in the lightness or saturation. The HSV components are defined as follows: H represents the colour type, S for the intensity, and **V represents the** brightness of the colour.

2) Since detecting perfect net corner locations has a level of uncertainty, manual adjustments improve real-world accuracy. Adding additional coordinates to the homograph matrix (H) could improve mapping accuracy. However, using any inaccurate point could increase the errors. An estimated centre of service line point ([409, 966]) was tested but led to an increased 2 cm error rate and thus should be disregarded.

3) Error rates increase for tennis balls closer to the camera due to landing point pixels no longer at the bottom. 4 linear difference equations were introduced at x and y coordinates, as shown in Figure 65, to readjust the tennis ball landing point position by using estimated landing point coordinates minus differences to change to the real landing point coordinate to correct this issue.

Due to the possibility of using the farthest landing point as the reference of estimated landing point equals the real landing point, it also introduces error rates if the ball location cannot be perfectly detected, thus using the centre landing point as a reference. Based on testing, among all 4 corners as shown in Figure 65, the right image, all fall within a 0.5 centimetre error rate. But if the landing point was significantly away from the five calculated reference points, the error rate could reach up to 10 cm. The primary cause of this inaccuracy was the use of the centre landing point as a reference, as the estimated centre landing point is also not the real landing point. Thus, the following modifications were implemented:

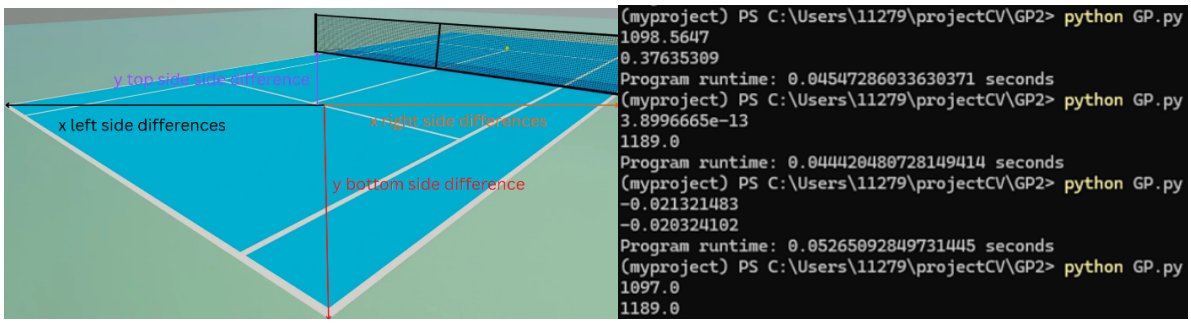



*Figure 65: Linear differences, ball detection results coordinates with real coordinates of the tennis court*

1. Use the net left corner landing point as the primary reference instead of the centre. 2. Eliminating closing operations to preserve ball centre accuracy, despite increasing noise and runtime. 3. Automating calculations using an optimised equation to refine the linear differences equation dynamically. Based on using images of the tennis ball manually placed at 4 corners similar to Figure 70, the program can automatically calculate 4 tennis ball landing point pixels compared to real corner pixels coordinates differences to set up the ratio equations. The code is shown below:

```
def newlandingpoint_calculation (left, right, bottom):
    global Top_left, Bottom_left, Top_right, Bottom_right, centery, above_centerx, below_centerx,
    below_centerx = (Top_left[1]-left[1]) / (Top_right[1]-left[1])
    above_centerx = (Bottom_right[1]-right[1]) / (Top_right[1]-right[1])
    centery = (right_most[0]-bottom[0]) / (net_left[0]-bottom[0])
if ballcenter[0] >= Top_right[1] :
    landingx, landingy = ballcenter[0]+above_centerx*(Top_right[1]-ballcenter[0]) , ballcenter[1] + radius +centery*np.log(np.exp(a))
else :
    landingx, landingy = ballcenter[0]+below_centerx*(Top_right[1]-ballcenter[0]) , ballcenter[1] + radius + centery*np.log(np.exp(a))
```

*Figure 66: Ratio function and real landing point calculation*

For further improvement, the y-axis distortion in the image frame can have a great impact that has been considered, thus introducing exponential form, using an additional ratio to adjust the original function to change the curve speed. Due to exponential growth, it is very difficult to adjust the ratio between 0.00009 and 0.00008. The soft plus [77] method is used to add a log to make it smoother, as for np.clip function is for limiting the area inside 701 to prevent overflow, using lower than exp(709)≈10^308 to avoid exceeding Python double precision limits. If result = 701, the system will alert the user ratio has caused overflow.

```
b=np.clip( ratioadjust*(ballbottom[0]-top[0]), None, 701)
centery = (bottom[0]-ballbottom[0]) / np.log(np.exp(b))
```

*Figure 67: Exponential function*

For ratio adjustment, after testing, using centre tennis ball landing point coordinates compared to the real landing point, adjusting the ratio to reach minimum difference is less efficient than, using left corner and right net corners difference summation, adjusting the ratio so that it reaches minimum difference in total. This is due to the centre landing point not being exact at the image centre, a perfect centre ratio will cost the leftmost having a higher y-axis pixels difference compared to the rightmost corner, meanwhile decreasing complexity for the user to place the tennis ball at the centre.

C) Real-world simulation

To validate the program's accuracy in real-world conditions, a real-world full sensor, sensor height, and lighting parameter adjustment are required. A GoPro Hero 13 camera was selected based on a 300 price, which is within budget, featuring: 1) 2.7K resolution at 240 fps. 2) Linear mode: 21 mm focal length with an 81.2-degree field of view.

Adjustments were based on reality considerations (human height 1.8 m, not perfectly covering hole ground with the greatest areas in the image, and lighting using a sky dome to simulate an indoor environment). To test overall performance, a comprehensive dataset shown in Figure 68 was used to cover the entire court and was manually evaluated.

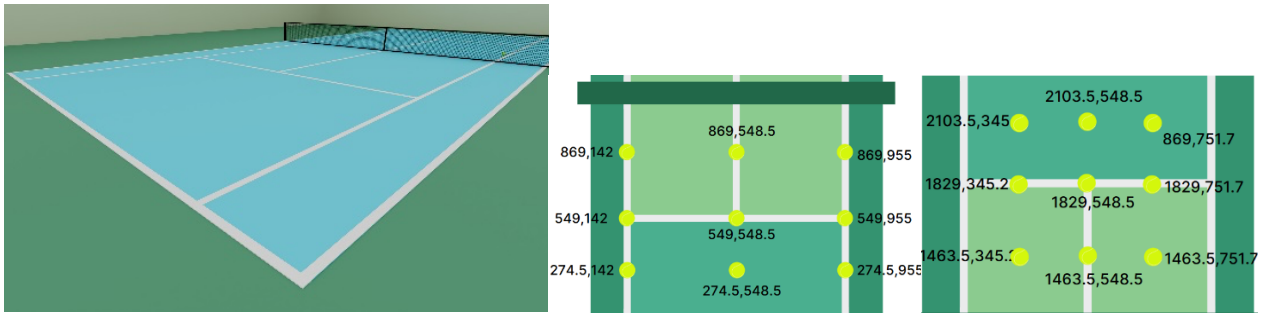



*Figure 68: Camera overview and comprehensive dataset for the alternative and current project*

### 9.1.4 Performance Analysis

A)Error rate analysis

In the current situation, using a 1.1 ratio yields the best results and error rate, with an average of 1.5 centimetres in tolerance shown in Figure 69.

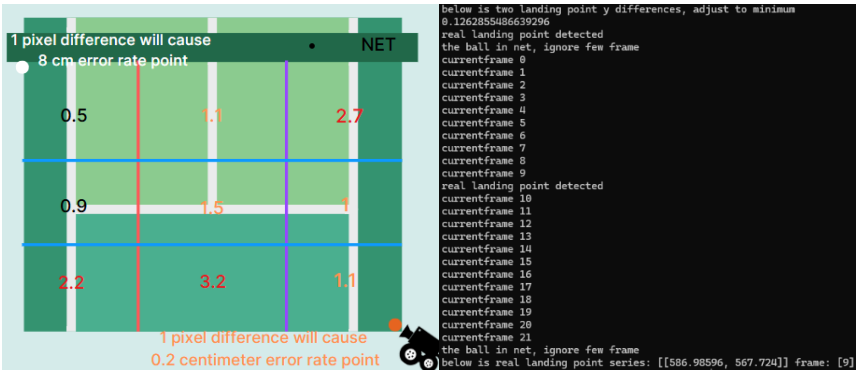



*Figure 69: Error rate map with error rate returned data*

Further improvements using exponential correction for x-coordinates were tested but yielded minimal benefits and significantly increased runtime, as the primary error source was y-axis distortion.

And to further reduce distortion, increasing the camera height to 3.5 meters could reduce distortion and estimate retained a maximum 2.7 cm error rate. And for the two camera settings shown in 9.2 C) accuracy, it will estimate a decrease of 27% error rate, but this program is targeted for efficiency, low costs of the two camera system; thus, both methods should be ignored.

As for real landing point selection analysis, using an image sequence of a tennis ball from the opponent side - player side – landing – rebounce - hit to the net, the result was correctly identified in the right image.

C) Runtime aspects analysis

Based on the following optimisation strategy: 1) Ignoring redundant circle area checks. 2) Utilising Opencv's optimised C++ hardware acceleration functions for computational efficiency. 3) Reducing local variable use.

The final result is around 0.0313 seconds, which means 30 frames per second. Due to it is very difficult to improve unless using other detection techniques, and this program is designed for video analysis, so it does not need to reach real-time computation, thus, 4 times latency for a 240 frame camera is good enough.

And for further Considerations, Frame-skipping methods could theoretically reduce computation by 8x if jumping 10-16 frames situation, but might introduce error rates and inconsistencies. If using a small jump, the re-computation will slow down the computation, thus cancelling the improvement. Due to the lack of a huge amount of real-world / simulation testing for all possible situations(time-consuming), frame-skipping is not recommended.

## 9.2 Current Program Detection Methodology (EL)

Consider 9.1 Alternative ground truth detection system that requires two extra professional sensors, which cannot be afforded in the current project for performance analysis, thus should change the strategy to use the existing 4 sensors to analyse performance.

A) Method selection

Similar to the 9.1 method, which focuses on the ground truth of the tennis ball landing point, four reference points (court left and right corner, net-left and right corners) can be identified to use for current project all sensors, and their corresponding 3D model coordinates can be used to compute the homograph matrix.

Subsequently, to mitigate the unpredictable pixel location of the actual landing point within the image issue, using image of tennis balls are placed at four corners (as shown in Figure 70, left image). The system will automatically adjust its landing points based on the initially detected four-corner pixels and compute a ratio function. This function provides the real landing point pixel location by shifting a few pixels left, right, up, or down based on the tennis ball's bottom pixel. Even if the tennis ball is not placed precisely at the corner position (Figure 70, right images), this method maintains a reliable landing point adjustment, although the result will increase 10 times due to placing over 5 cm.

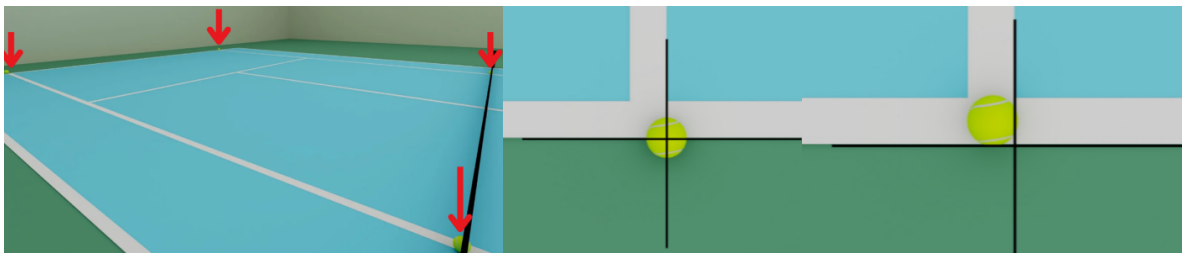


*Figure 70: Tennis balls placed in four corners situation*

To mitigate error rate further, since two sensors cover each player's ground, each landing point will have two estimated ground truth coordinates calculated from two sensor frames. Averaging these values provides a more accurate ground truth estimation for the landing point.

B) Code implementation

For code implementation, based on the alternative ground truth detection system implementation, a few adjustments in the code have been considered. Firstly, due to sensor placement and a single-player scenario in which court boundaries are at single-sidelines, the program is enhanced by leveraging ball location data to create a bounding box and filtering out all white pixels outside this region. This approach removes boundary lines while preserving the single-player boundary shown in Figure 71. left image.

Next, following by similar net corner detection approach of 9.1, sample points are extracted from two boundary lines simply using the same column y-axis maximum and minimum white pixels, allowing for the creation of two linear equations to determine the left and right side corners. Due to the unpredictability of net corners, it is best based on the result manually adjust. Once finding corners like shown in Figure 71, the right image, can easily use the corner ratio program to adjust to the best result.

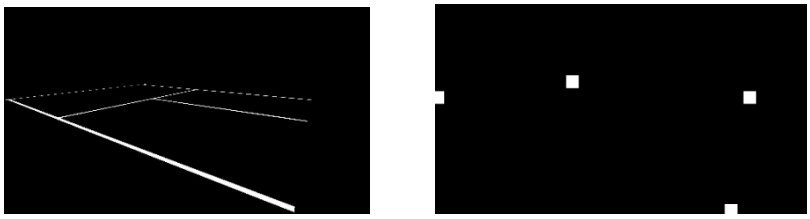


*Figure 71: Tennis court lines and the 4 corners location*

C) Accuracy Testing

Due to in real-world circumstances testing lack of professional sensors and placing the tennis ball manually will introduce error rate, based on software to simulate reality will be best option to testing ground truth error rate of this method, using above program with 9.1 similar sensor setting of 4 GoPro Hero 13 cameras, featuring: 1] 2.7K resolution at 240 fps. 2] Linear mode: 21 mm focal length with an 81.2-degree field of view [78]. The simulation sensor image is similar to Figure 70, and a comprehensive dataset, closely aligned with Figure 68 dataset, was utilised to test the system.

After calculating each court sector's error rate, the visualisation of a single right-side sensor's return data is illustrated in Figure 72. left side, consider 9.6 cm error rate is caused by distortion, x exponential form is introduced, but still cannot decrease the error rate. On the other hand, consider placing tennis balls at the 4 corners landing point; the procedure does not always result in an exact corner placement. Placing far away from the corner over 5 centimetres can result in the data in the middle image. Additionally, after averaging the data from both cameras, the final error rate is presented on the right side image. The results indicate that the tennis ball location closer to the camera results in lower error rates, whereas increased distance introduces greater distortion. Averaging the data from two cameras effectively cancels out distortions.

In summary, for a single-camera setup, placing 4 tennis balls at the corner stage introduces a 5 cm human error rate situation, significantly increases the error rate, potentially reaching an estimated increase of 142.02%. However, utilising a dual-camera setup and positioning the ball near the corner reduces the error rate by **27.07%**. In real-world applications, implementing a dual-camera system and ensuring tennis balls are placed near the corner is crucial.

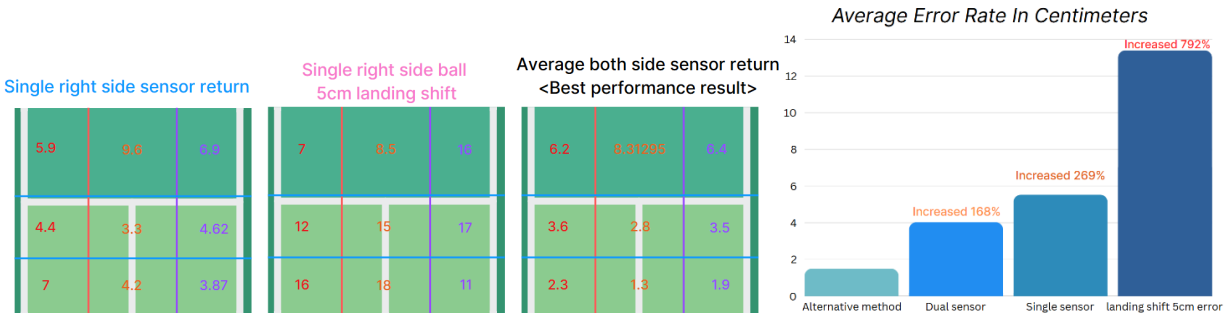



*Figure 72: Tennis court lines and the 4 corners location*

## 9.3 Game Logic Scoring Implementation(EL)

Based on the returned valid landing point sequence, using tennis court property to set different landing zones, and using tennis game rules, a program could be set to check the current game state and scoring system. As shown in Figure 73. left image top, based on landing zones and variable(player) to represent which player is hitting the ball, could check whether the player serves or rallies successfully. Then, using the returned value to set the current scoring system in the right image. And finally decide which player wins the round, or the entire game. For testing, using the dataset in the left image bottom, to simulate different situations, and the system successfully handled it.

```python
def serve_fault(player, landing_zone):
    valid_zones = {1: [4, 5], 2: [1, 2]}  # Valid zones for Player 1 and Player 2
    serve_side = points[player] % 2
    required_zone = valid_zones[player][serve_side]
    return landing_zone != required_zone


def rally_valid(player, landing_zone):
    if player == 1:
        return 4 <= landing_zone <= 6  # Ball must land on Player 2's side
    else:
        return 1 <= landing_zone <= 3  # Ball must land on Player 1's side

#Testing Dataset sequence
game(15, 5)  # Serve fault
game(15, 5)  # Double fault, Player 2 gains a point

game(600, 1500, returned=True)  # Valid serve, Player 2 returns
game(300, 1000, returned=True)  # Rally continues
game(600, 1600, returned=False)  # Player 2 fails to return, Player 1 gains a point

game(400, 1400, returned=False)  # Player 1 serves, valid shot

game(600, 1500, returned=True)  # Player 1 serves, valid shot
game(30, 250, returned=True)  # Rally continues
game(800, 1800, returned=False)  # Player 2 fails to return, Player 1 gains a point

game(400, 1400, returned=True)  # Player 1 serves, valid shot
game(300, 1000, returned=True)  # Rally continues
game(200, 1200, returned=False)  # Player 2 fails to return, Player 1 gains a point
```

```python
def game(x, y, returned=True):
    global player_shot, state
    landing_zone = determine_landing_zone(x, y)
    print(f'Player {player_shot} lands in {landing_zone}')

    # Serve phase
    if state in (1, 2):
        if serve_fault(player_shot, landing_zone):
            if state == 2:
                update_score(3 - player_shot)
            else:
                state = 2  # Second serve
            return
        if not returned:
            update_score(player_shot)
        else:
            state = 0
            ball_landing_zones[player_shot].append(landing_zone)
            player_shot = 3 - player_shot
        return

    # Rally phase
    if state == 0:
        if not returned:
            update_score(player_shot)
            return
        if not rally_valid(player_shot, landing_zone):
            update_score(3 - player_shot)  # Opponent gains a point
            return
        ball_landing_zones[player_shot].append(landing_zone)
        player_shot = 3 - player_shot  # Switch players
```

*Figure 73: Game logic, game state and testing data set.*

## 9.4 Additional Research Conclusion (EL)

For conclusion, Additional Research provides two efficient, systematic, 3.2 and 8.3 cm guarantee maximum error rate methods, and in addition to supporting the project, builds a system which measures the ground truth of the tennis ball landing point with the support of game logic and scoring system to track the current game state.

Due to using existing sensor calculated ground truth data with four cameras in total, it has twice the calculation runtime and increased 168.78% error rate compared to the alternative method; thus, for accurate ground truth measurement is best to use the alternative ground truth detection system.

On the other hand, because of the target change from using a professional GoPro camera to an iPhone system due to unexpected resistance, it is very sad that the system cannot be used for the current project's performance analysis, due to both systems are unsuitable for the current project. Even though both iPhone and camera have 240 fps, the differences in sensor size and compression of image cause GoPro to have an advantage in performance in clarity, motion blur of the ball and other areas, which iPhone system has poor performance when handling sudden deceleration under high contrast and bright scenes. Even in slow motion, the ball will look smeared and not fully round, whereas professional sensors have a contrast result. Although in reality cannot fully test the system, the simulation results, which are close to reality, have provided a fully tested, guaranteed performance proof. Thus, the system can be used directly in reality with small error rate increases because of unpredictability.

# 10 Acknowledgements

We gratefully acknowledge the guidance and support of our supervisor, Dr. Dimitris Agrafiotis, whose insights and encouragement were invaluable throughout this work.

[1] Tennisnerd, “Prize money.” Available: https://www.tennisnerd.net/prize-money/

[2] Wikipedia, “Hawk-Eye.” Available: https://en.wikipedia.org/wiki/Hawk-Eye

[3] CNBC, “How Sony’s Hawk-Eye works at the US Open,” 9 Sep. 2023. Available: https://www.cnbc.com/2023/09/09/how-sonys-hawk-eye-works-at-the-us-open.html

[4] Wikipedia, “Fastest recorded tennis serves.” Available: https://en.wikipedia.org/wiki/Fastest_recorded_tennis_serves

[5] J. Hsu, “Cameras, fouls and referees,” Popular Mechanics, 5 Apr. 2011. Available: https://www.popularmechanics.com/adventure/sports/a5772/cameras-fouls-and-referees/

[6] J. Hsu, “Cameras, fouls and referees,” Popular Mechanics, 5 Apr. 2011. Available: https://www.popularmechanics.com/adventure/sports/a5772/cameras-fouls-and-referees/

[7] TennisLeo, “Best tennis line-calling and smart net-post systems.” Available: https://www.tennisleo.com/best-tennis-line-calling-and-smart-net-post-systems/

[8] The Times of India, “Hawk-Eye: a decade of accuracy,” 13 Jan. 2016. Available: https://timesofindia.indiatimes.com/top-stories/hawk-eye-a-decade-of-accuracy/articleshow/50534732.cms

[9] Hawk-Eye Innovations, *Hawk-Eye in Tennis*, 2015.

[10] InOut Tennis, “In/Out tennis system.” Available: https://inout.tennis/en/index.htm

[11] Baseline Vision, “Baseline Vision product.” Available: https://www.baselinevision.com/product

[12] FOXTENN, “In & Out system.” Available: http://www.foxtenn.com/in&out

[13] Stanford University, “Final Report CS231A, 2016.” Available: https://web.stanford.edu/class/cs231a/prev_projects_2016/final_report_v2.pdf

[14] Stanford University, “Final Report EE367, Winter 2018.” Available: https://web.stanford.edu/class/ee367/Winter2018/fazio_fisher_fujinami_ee367_win18_report.pdf

[15] S. A. R. Szeliski, “A taxonomy and evaluation of dense two-frame stereo correspondence algorithms,” *International Journal of Computer Vision*, Apr. 2002. doi: 10.1023/A:1014573219977

[16] Luxonis, “OAK-D LR: Long Range Stereo Camera,” 2023. Available: https://docs.luxonis.com/projects/hardware/en/latest/pages/OAK-D-LR/

[17] Luxonis, “OAK-D S2: Compact and Modular Stereo Depth + AI Camera,” 2023. Available: https://docs.luxonis.com/projects/hardware/en/latest/pages/OAK-D-S2/

[18] R. Szeliski, *Computer Vision: Algorithms and Applications*, 2nd ed., London, U.K.: Springer, 2022.

[19] R. Klette, *Concise Computer Vision: An Introduction into Theory and Algorithms*, London, U.K.: Springer, 2014.

[20] Q. Y., “How fast does a tennis ball go?” TheRacketSports, 2025. Available: https://theracketsports.com/how-fast-does-a-tennis-ball-go/

[21] Elsevier, “A study on object detection.” Available: https://www.sciencedirect.com/science/article/pii/S1877050922001363

[22] S. Bogireddy, “Comparative study of background subtraction algorithms using OpenCV,” Medium. Available: https://medium.com/@srinivas.bogireddy/comparative-study-of-background-subtraction-algorithm-for-moving-object-detection-using-opencv-e5349bbe33fa

[23] Springer, “Study on background modeling.” Available: https://link.springer.com/article/10.1007/s42452-019-1356-9

[24] Elsevier, “Background subtraction techniques.” Available: https://www.sciencedirect.com/science/article/pii/S0167865505003521

[25] Springer, “Optical flow evaluation.” Available: https://link.springer.com/article/10.1023/B:VISI.0000011205.11775.fd

[26] Baeldung, “Optical Flow: Lucas-Kanade method.” Available: https://www.baeldung.com/cs/optical-flow-lucas-kanade-method

[27] Optica, “Optical Flow accuracy study.” Available: https://opg.optica.org/ao/fulltext.cfm?uri=ao-63-3-831&id=545651

[28] ResearchGate, “Ball recognition using Circle Hough Transform and Neural Classifier.” Available: https://www.researchgate.net/publication/220601357_A_new_algorithm_for_ball_recognition_using_circle_Hough_transform_and_neural_classifier

[29] ResearchGate, “Deep learning-based tennis ball recognition.” Available: https://www.researchgate.net/publication/365800010_Deep_Learning-Based_Algorithm_for_Recognizing_Tennis_Balls

[30] Stanford University, “Final Report CS231A, 2016.” Available: https://web.stanford.edu/class/cs231a/prev_projects_2016/final_report_v2.pdf

[31] Óbuda University, “Object detection with YOLO.” Available: https://conf.uni-obuda.hu/sisy2004/Vamossy.pdf

[32] ResearchGate, “Comparison of YOLOv5 and YOLOv8 for mobile UI detection.” Available: https://www.researchgate.net/publication/372873844_A_Comparison_of_YOLOv5_and_YOLOv8_in_the_Context_of_Mobile_UI_Detection

[33] IEEE, “Skeleton tracking with SkeleTrack.” Available: https://ieeexplore.ieee.org/abstract/document/10756983

[34] Queen’s University, “SwimPoser Project thesis.” Available: http://hdl.handle.net/1974/31763

[35] C. et al., “Energy Optimized YOLO: Quantized inference for real-time edge AI object detection,” 2025.

[36] OpenCV Documentation, “Calibration methods.” Available: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html

[37] OpenCV Documentation, “Triangulation methods.” Available: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html#ga357634492a94efe8858d0ce1509da869

[38] ResearchGate, “Stereo-digital image correlation system calibration.” Available: https://www.researchgate.net/publication/340989088_Calibrating_stereo-digital_image_correlation_system_using_synthetic_speckle-pattern_calibration_target

[39] BasicAI, “Camera calibration: Intrinsic, extrinsic, and distortion.” Available: https://docs.basic.ai/docs/camera-intrinsic-extrinsic-and-distortion-in-camera-calibration

[40] OpenCV Documentation, “Camera calibration basics.” Available: https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga3207604e4b1a1758aa66acb6ed5aa65d

[41] Dirk Farin, “Ball tracking for sports,” 2004. Available: https://dirk-farin.net/publications/data/Farin2004b.pdf

[42] OpenCV Documentation, “Stereo calibration example.” Available: https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga91018d80e2a93ade37539f01e6f07de5

[43] Dirk Farin, “Ball tracking for sports,” 2004. Available: https://dirk-farin.net/publications/data/Farin2004b.pdf

[44] Wikipedia, “Rigid transformation.” Available: https://en.wikipedia.org/wiki/Rigid_transformation

[45] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, Cambridge University Press, 2nd ed., 2004. Available: https://www.r-5.org/files/books/computers/algo-list/image-processing/vision/Richard_Hartley_Andrew_Zisserman-Multiple_View_Geometry_in_Computer_Vision-EN.pdf

[46] University of Illinois, “Lecture: Structure from Motion (CS543),” Spring 2017. Available: https://courses.grainger.illinois.edu/cs543/sp2017/lectures/Lecture15-Structure-from-Motion-Vision_Spring2017.pdf

[47] University of Illinois, “Lecture: Structure from Motion (CS543),” Spring 2017. Available: https://courses.grainger.illinois.edu/cs543/sp2017/lectures/Lecture15-Structure-from-Motion-Vision_Spring2017.pdf

[48] OpenCV Documentation, “Triangulation methods.” Available: https://docs.opencv.org/3.4/d0/dbd/group__triangulation.html

[49] IEEE, “Stereo vision systems study,” 2019. Available: https://ieeexplore.ieee.org/document/8864723

[50] Wikipedia, “Savitzky–Golay filter.” Available: https://en.wikipedia.org/wiki/Savitzky–Golay_filter

[51] Springer, “Signal smoothing with Savitzky–Golay filters.” Available: https://link.springer.com/article/10.1007/s12283-013-0144-9

[52] RIT, “Baseball trajectory simulation.” Available: http://spiff.rit.edu/richmond/baseball/traj/traj.html

[53] Durham University, “Physics lecture: Trajectory modeling.” Available: https://astro.dur.ac.uk/~tt/MSc/Lecture2.pdf

[54] J. Mehring, J. H. Eden, K. Müller, and P. König, “Forward and inverse models for trajectory prediction and imitation,” *Neural Networks*, vol. 21, no. 5, pp. 618–627, 2008. doi: 10.1016/j.neunet.2008.03.001

[55] J. Bagnall, A. Lines, A. Bostrom, J. Large, and E. Keogh, “The great time-series classification bake-off,” *Data Mining and Knowledge Discovery*, vol. 31, no. 3, pp. 606–660, May 2017

[56] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” *Advances in Neural Information Processing Systems*, vol. 27, Dec. 2014. Available: https://arxiv.org/abs/1409.3215

[57] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, 3rd ed., Pearson, 2009

[58] K. Cho et al., “Learning phrase representations using RNN Encoder–Decoder for statistical machine translation,” *Proceedings of EMNLP*, 2014, pp. 1724–1734. Available: https://arxiv.org/abs/1406.1078

[59] C. Walker, D. Doersch, A. Gupta, and M. Hebert, “An uncertain future: Forecasting from static images using variational autoencoders,” *European Conference on Computer Vision (ECCV)*, 2016, pp. 835–851

[60] H. Sakoe and S. Chiba, “Dynamic programming algorithm optimization for spoken word recognition,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 26, no. 1, pp. 43–49, Feb. 1978. doi: 10.1109/TASSP.1978.1163055

[61] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. doi: 10.1162/neco.1997.9.8.1735

[62] S. Bai, J. Z. Kolter, and V. Koltun, “An empirical evaluation of generic convolutional and recurrent networks for sequence modeling,” *arXiv preprint arXiv:1803.01271*, 2018. Available: https://arxiv.org/abs/1803.01271

[63] M. Schuster and K. K. Paliwal, “Bidirectional recurrent neural networks,” *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, Nov. 1997. doi: 10.1109/78.650093

[64] Allen Institute for AI, “Figure 3-1,” 8 Aug. 2017. Available: http://ai2-s2-public.s3.amazonaws.com/figures/2017-08-08/f7bdb849dafe17c952bfd88b879e01f74cf59d78/4-Figure3-1.png

[65] T.-Y. Lin et al., “Microsoft COCO: Common objects in context,” *European Conference on Computer Vision (ECCV)*, 2014, pp. 740–755. Available: https://arxiv.org/abs/1405.0312

[66] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding,” *arXiv preprint arXiv:1510.00149*, 2015. Available: https://arxiv.org/abs/1510.00149

[67] Microsoft, “ONNX Runtime: Cross-platform, high-performance scoring engine for ML models,” 2023. Available: https://onnxruntime.ai

[68] B. Jacob et al., “Quantization and training of neural networks for efficient integer-arithmetic-only inference,” *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 2704–2713

[69] Streamlit, “The fastest way to build data apps,” 2023. Available: https://streamlit.io

[70] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, “Optuna: A next-generation hyperparameter optimization framework,” *Proceedings of the 25th ACM SIGKDD*, 2019, pp. 2623–2631. doi: 10.1145/3292500.3330701

[71] A. Vaswani et al., “Attention is all you need,” *Advances in Neural Information Processing Systems*, vol. 30, 2017. Available: https://arxiv.org/abs/1706.03762

[72] Wikipedia, “Homography (computer vision).” Available: https://en.wikipedia.org/wiki/Homography_(computer_vision)

[73] OpenCV Documentation, “Morphological transformations.” Available: https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html

[74] OpenCV Documentation, “Finding contours.” Available: https://docs.opencv.org/3.4/df/d0d/tutorial_find_contours.html

[75] OpenCV Documentation, “Minimum enclosing circle.” Available: https://docs.opencv.org/3.4/dd/d49/tutorial_py_contour_features.html

[76] OpenCV Documentation, “Point polygon test.” Available: https://docs.opencv.org/3.4/dc/d48/tutorial_point_polygon_test.html

[77] Wikipedia, “Softplus.” Available: https://en.wikipedia.org/wiki/Softplus

[78] GoPro, “HERO13 Black: Video settings and resolutions.” Available: https://community.gopro.com/s/article/HERO13-Black-Video-Settings-And-Resolutions?language=en_US