Stellenbosch
UNIVERSITY
IYUNIVESITHI
UNIVERSITEIT

forward together
sonke siya phambili
saam vorentoe

MECHATRONIC PROJECT 478
FINAL REPORT

Combining object tracking algorithms with probabilistic graphical models for accurate ball localization

Author:
Alex Thorpe

Student Number:
25934171

October 19, 2025

Supervisor: Dr. C. E. Van Daalen

Plagiarism Declaration

I have read and understand the Stellenbosch University Policy on Plagiarism and the definitions of plagiarism and self-plagiarism contained in the Policy [Plagiarism: The use of the ideas or material of others without acknowledgement, or the re-use of one's own previously evaluated or published material without acknowledgement or indication thereof (self-plagiarism or text-recycling)].

I also understand that direct translations are plagiarism, unless accompanied by an appropriate acknowledgement of the source. I also know that verbatim copy that has not been explicitly indicated as such, is plagiarism.

I know that plagiarism is a punishable offence and may be referred to the University's Central Disciplinary Committee (CDC) who has the authority to expel me for such an offence.

I know that plagiarism is harmful for the academic environment and that it has a negative impact on any profession.

Accordingly all quotations and contributions from any source whatsoever (including the internet) have been cited fully (acknowledged); further, all verbatim copies have been expressly indicated as such (e.g. through quotation marks) and the sources are cited fully.

I declare that, except where a source has been cited, the work contained in this assignment is my own work and that I have not previously (in its entirety or in part) submitted it for grading in this module/assignment or another module/assignment. I declare that have not allowed, and will not allow, anyone to use my work (in paper, graphics, electronic, verbal or any other format) with the intention of passing it off as his/her own work.

I know that a mark of zero may be awarded to assignments with plagiarism and also that no opportunity be given to submit an improved assignment.

[illegible]	25934171
Signature	Student number
Alex Thorpe	October 19, 2025
Initials and surname	Date

Executive Summary

Title of Project
Combining Object Tracking Algorithms with Probabilistic Graphical Models for Accurate Ball Localization
Objectives
This project aims to estimate a ball's position in real time using standard video footage. Multiple object tracking algorithms will be applied, and their outputs fused using a probabilistic graphical model (PGM) to enhance accuracy, robustness, and responsiveness. The final goal is to evaluate the PGM-based fusion model against individual trackers.
What is current practice and what are its limitations?
Hawk-Eye and Tracab are industry leaders in ball tracking and sports officiating. Hawk-Eye supports over 20 sports with real-time tracking and visualization, while Tracab focuses on football, analyzing player movement and ball trajectory (Hawk-Eye Innovations 2024; Tracab 2024). Despite their accuracy, both systems require complex, expensive multi-camera setups and proprietary infrastructure, limiting their use in smaller-scale, research, or open-source applications.
What is new in this project?
This project introduces an accurate, low-complexity ball tracking system that requires minimal camera infrastructure, making it accessible for research, amateur sports, and low-budget environments.
If the project is successful, how will it make a difference?
If successful, this project could enhance decision-making across various sports by accurately determining ball location in real time — such as confirming goals or out-of-bounds events — making advanced tracking more accessible beyond elite-level systems.
What are the risks to the project being a success? Why is it expected to be successful?
A key risk is that the system may not match the efficiency or accuracy of existing commercial trackers, limiting its ability to fully replace them. However, by studying and building on established tracking methods, this project is expected to deliver a more lightweight and accessible alternative — one that balances performance with lower complexity and cost.
What contributions have/will other students made/make?
N/A
Which aspects of the project will carry on after completion and why?
After completion, the project could be extended by incorporating sport-specific rules, enabling the system to automatically detect rule violations such as offside positions, fouls, or out-of-bounds events. This would build on the core tracking functionality and move the system toward real-time decision support and automated officiating.

What arrangements have been/will be made to expedite continuation?
All components of the project will be thoroughly documented, with clear definitions of the system architecture, algorithms, and data handling. This will ensure the project can be easily continued by future students or refined further for potential commercialization.

Evaluation of ECSA Exit Level Outcomes

ECSA Outcomes Assessed in this Module	
ECSA Outcome	**Addressed**
GA 1. Problem solving: Demonstrate competence to identify, assess, formulate and solve convergent and divergent engineering problems creatively and innovatively.	Throughout the literature review process and figuring out the algorithms.
GA 2. Application of scientific and engineering knowledge: Demonstrate competence to apply knowledge of mathematics, basic science and engineering sciences from first principles to solve engineering problems.	During the experimentation and testing portion of the project and when applying the algorithms.
GA 3. Engineering Design: Demonstrate competence to perform creative, procedural and non-procedural design and synthesis of components, systems, engineering works, products or processes.	During the concept design and algorithm decision.
GA 5. Engineering methods, skills and tools, including Information Technology: Demonstrate competence to use appropriate engineering methods, skills and tools, including those based on information technology.	Throughout the project the use of different coding methods will be used and learning new skills such as PGMs.
GA 6. Professional and technical communication: Demonstrate competence to communicate effectively, both orally and in writing, with engineering audiences and the community at large.	Project Proposal; Progress Report; Preliminary Draft; Final Report; Oral Presentation; Project Poster.

GA 8. Individual, Team and Multidisciplinary Working: Demonstrate competence to work effectively as an individual, in teams and in multidisciplinary environments.	This project shows individual work very clearly.
GA 9. Independent Learning Ability: Demonstrate competence to engage in independent learning through well-developed learning skills.	Will have to self-learn many different complex topics.

Contents

List of Figures

List of Tables

List of Abbreviations

ADF Assumed Density Filtering

BGR Blue Green Red (colour space)

BN Bayesian Network

BP Belief Propagation

BU Belief Update

CLG Conditional Linear Gaussian

CNN Convolution Neural Network

CPD Conditional Probability Distribution

CV Computer Vision

DL Deep Learning

EP Expectation Propagation

FPS Frames Per Second

HSV Hue Saturation Value (colour space)

IPA Image Processing Algorithm

ML Machine Learning

MN Markov Network

PGM Probabilistic Graphical Model

RV Random Variable

YOLO You Only Look Once

List of Symbols

$P(A)$	Probability of event A occurring
$P(A \mid B)$	Conditional probability of event A given event B
$P(X_1, \ldots, X_n)$	Joint probability distribution over variables X_1 through X_n
Σ	Summation operation

1 Introduction

The growing importance of data-driven decision-making in sports has accelerated the adoption of advanced technologies such as Machine Learning (ML), computer vision, and real-time analytics. In particular, ball tracking has become a critical component in modern sports for enhancing player performance analysis, enabling data-supported officiating, and facilitating targeted training interventions. High-profile systems such as Hawk-Eye and TRACAB have become industry standards, Hawk-Eye being widely used for adjudication in tennis and cricket (Hawk-Eye Innovations 2024), while TRACAB supports tactical and performance analysis in football and other team sports (Tracab 2024).

Despite their effectiveness, these systems depend on dense camera arrays and high-performance computing infrastructure installed around stadiums, resulting in high operational costs. Consequently, their deployment is largely limited to professional leagues and elite-level clubs. This creates a gap for smaller leagues, grassroots sports organizations, and researchers who require tracking tools but lack access to such costly resources.

This project investigates a low-cost, accessible solution for real-time ball tracking using pre-recorded video footage sourced from online platforms. By implementing and comparing multiple lightweight tracking algorithms—including classical computer vision techniques and pre-trained object detectors such as YOLO—the project aims to develop a robust pipeline capable of estimating the ball's position. A key innovation in the proposed framework is the fusion of uncertain outputs from multiple Image Processing Algorithm (IPA) using a Probabilistic Graphical Model (PGM), which enables robust inference even in the presence of noise and partial information (Koller and Friedman 2009). This probabilistic fusion strategy helps mitigate the limitations of individual trackers and reduces overall tracking errors, particularly in challenging conditions such as occlusion and motion blur.

The objective is to develop a cost-effective and computationally efficient tracking system that can run on consumer-grade hardware without compromising significantly on accuracy. This contributes to the democratization of sports analytics, opening the door to broader adoption in non-professional environments such as university clubs, school competitions, and community sports settings.

This research is conducted by Mr. A.H. Thorpe as part of the Mechatronic Project 478/488 under the supervision of Dr. Corne Van Daalen. It aligns with the graduate attributes prescribed by the Engineering Council of South Africa (ECSA).

1.1 Motivation

In many sports, decisions made by officials are often prone to error because they rely heavily on human judgment. These errors can accumulate, with some leagues averaging up to five officiating mistakes per match (Daily News 2023). For example, in the Premier League, referees still rely on manually drawing lines on video footage to determine ball position for offside or goal-line decisions (Premier League 2020). While this method can be effective, it remains susceptible to inaccuracies caused by varying camera angles and inherent human limitations. An accurate, real-time ball tracking system could significantly improve decision-making by providing objective and precise positional data, reducing inconsistencies during match officiating (The Guardian 2025).

Beyond officiating, precise ball tracking also offers substantial benefits in player training and performance analysis. By capturing various aspects of both player movement and ball dynamics, coaches and athletes can gain deeper insights into performance metrics, enabling targeted improvements in technique, decision-making, and strategy. Such data-driven feedback has been shown to enhance skill development and game understanding across multiple sports disciplines (Oliva-Lozano et al. 2022).

In addition to these high-level benefits, many smaller clubs, schools, and amateur leagues lack the resources to implement advanced commercial tracking systems such as Hawk-Eye or TRACAB. These systems require high-performance cameras and computing infrastructure, making them inaccessible to most grassroots or academic environments. There is a clear need for a low-cost, computationally efficient alternative that can run on widely available consumer-grade hardware.

To address this gap, the project proposes a lightweight video-based tracking framework capable of estimating the ball's position in three-dimensional space using multiple complementary detection techniques. By combining the strengths of different methods, the system aims to deliver reliable tracking performance even under challenging conditions such as occlusion or motion blur.

1.2 Objectives

The objective of this project is to develop a model that can accurately and reliably estimate the position of a ball in real time using pre-recorded video footage. This will be achieved through the fusion of multiple IPA, each capable of independently detecting the ball's position. The outputs of these algorithms will be combined using a PGM to improve robustness and reduce the estimation error in position. The following objectives define the scope of this project:

1. Develop and implement multiple IPA to estimate the ball's position from video footage.
2. Design a PGM-based fusion method to combine the outputs of individual algorithms, improving tracking accuracy and reducing errors from isolated failures.

3. Evaluate and optimize the system to use the minimal number of algorithms required while maintaining acceptable performance.

4. Compare the final model's performance against each individual algorithm in terms of accuracy, robustness, and computational efficiency.

1.3 Report Overview

This section outlines each section that will be discussed in this report. The idea gets formed and the objectives set in Chapter 1. Chapter 2 focuses on the background and literature of the project where core concepts get formed from existing work. From there the Model Design takes place in Chapter 3, this is where the requirements of the IPAs and PGMs are set. Chapter 4 will then go into the implementation of the IPAs and PGM and how the final Algorithms are choosen. Chapter 5 then goes into Evaluation and how the various parameters for the IPA and PGM were selected. Finally the last chapter will cover the conclusion of the report.

2 Background and Literature Review

Object tracking is a task that dates back to the earliest days of human civilization, from following animal tracks to observing celestial bodies for navigation. As technology has advanced, computers have become a key tool in tracking objects with high precision. One of the most common examples in modern life is the Global Positioning System (GPS), which allows users to determine their position anywhere on Earth (Challa 2011).

A major industry that relies heavily on object tracking is sports. Companies such as Hawk-Eye and TRACAB are global leaders in this domain, with Hawk-Eye specializing in officiating decisions and TRACAB focusing on performance analysis and training (Tracab 2024; Hawk-Eye Innovations 2024). Accurate tracking in sports benefits a wide range of stakeholders — from fans and broadcasters to coaches, players, and officials (Labayen et al. 2014). Given that sports are governed by clear rules and boundaries, it is natural that computer-aided systems are increasingly being used to enhance decision-making on and off the field.

Despite its advantages, ball tracking in sports presents numerous challenges. Balls tend to be small, fast-moving objects, often affected by motion blur and frequent occlusion in video footage. These issues can severely limit the performance of traditional image processing techniques. Furthermore, the nature of the tracking problem can vary significantly between sports. For instance, in table tennis, the ball exhibits highly nonlinear and non-periodic motion, making it particularly difficult to track reliably (Kamble, Keskar, and Bhurchandi 2019).

Because of these challenges, no single tracking algorithm can perform robustly in all scenarios. Each technique has its own weaknesses depending on the scene, motion, and visual obstructions. This motivates the use of Probabilistic Graphical Models, which provide a principled framework for combining uncertain outputs from multiple algorithms (Koller and Friedman 2009). PGMs allow relationships between multiple random variables to be explicitly modeled, making them particularly useful for data fusion and robust inference under uncertainty.

This section provides a review of the core technical concepts and prior work relevant to this project. It includes discussions on existing commercial tracking systems, Computer Vision techniques for tracking, IPA, PGMs and, identifies the current gaps in the literature that this project aims to address.

2.1 Existing Commercial Tracking systems

As mentioned in the introduction, two of the leading commercial solutions for ball tracking in sports are Hawk-Eye and TRACAB. Hawk-Eye operates using an array of high-speed cameras positioned around the stadium to capture the ball's trajectory in real time with sub-millimetre accuracy (TennisNerd 2024). This system, while highly accurate, is prohibitively expensive for widespread use. Installation costs range between R1 million and R1.25 million per court (approximately \$55,000–\$70,000 USD), making it inaccessible to amateur clubs and individuals (Wong 2016). In addition to its cost, Hawk-Eye requires complex infrastructure, including camera placements two to three stories above the field, which adds to the installation difficulty and venue constraints.

TRACAB, another global leader, combines camera-based tracking with sensor-based solutions. The system includes multiple high-speed cameras—often sourced from Hawk-Eye—alongside a sensor chip embedded in the ball, developed by the German company Cairos Technology AG (World Intellectual Property Organization 2023). TRACAB is commonly used for both training applications and goal-line decision-making. However, its cost is even higher, with installation reported at around R5 million (approx. \$270,000 USD) and an additional R70,000 (approx. \$3,800 USD) per game to operate the system (Wiebe 2013).

While these high-end systems are effective, they are far beyond the reach of most local sports clubs, schools, and individual athletes. More affordable alternatives are emerging, such as XBotGo, which offers a ball-tracking system for a once-off cost of approximately R6,300 (about \$340 USD) (XbotGo 2024). Although this represents a more accessible option, it still requires the user to provide their own compatible camera equipment, which contributes to the total cost. Additionally, shipping limitations may restrict access to certain regions, particularly in developing countries.

2.2 Computer Vision techniques for tracking

Computer Vision is used when significant information is needed to be extracted from images (Stockman and Shapiro 2001). In terms of tracking objects this is vital, as this opens the door to be able to extract vital information of where the object is located. The foundation of object tracking is object detection, which is identifying an object of interest in an image. Wheres object tracking evolved into is being able to "see" the motion of the object as it moves through the frames of a video(Zhao et al. 2015). Object tracking doesn't come without its challenges with the most common being occlusion, illumination variation, and fast motion(Soleimanitaleb, Keyvanrad, and Jafari 2019).

2.2.1 Traditional Tracking and Detecting methods

There are many different methods that lead to how CV was used for tracking but most of them are based on foundational mathematics and statistics.

One such example uses recursive Bayesian logic formulated and derived from the Bayesian probabilistic framework. Examples of algorithms traditionally found are the Kalman filter, extended Kalman filter, unscented Kalman filter, point mass filter, and particle filter. These algorithms were developed to find a resolution to generic estimation and filtering problems (Challa 2011).

An example based on using mathematical models for detection and tracking is from a study done in 2013 that proposed an algorithm based on integrating the mechanisms of the Human Visual System (HVS) where it detected and tracked infrared dim and a small target. The method is based on 4 steps, firstly a multi-scale Difference of Gaussians, which is a method of detecting features of different scales, which is then placed in feature maps at different scales, the next step was to add a Gaussian window to this map at a location near the object, the penultimate step was to normalize the image and decide on the location of the object in this frame and if this is the final step the algorithm stops. However if there is another frame it estimates the location on the next frame via a Proportional-Integral-Derivative algorithm and goes back to step 1(Mirzaei et al. 2023; Dong et al. 2014).

These both provide an idea of what is traditionally done in terms of tracking and detecting objects through CV.

2.2.2 Modern Deep Learning-Based Tracking

As technology in software and hardware advanced so did the methods in which tracking and detecting was done. Convolution Neural Networks based on Deep Learning technology(Pal et al. 2021), is a relatively recent breakthrough in CV and how tracking is done from video footage.

CNNs are used to identify features in which they are used to localize and classify the objects within the frames(Pal et al. 2021), these make it a very powerful tool for tracking and detecting objects. There are two methods, being discriminative and generative. The discriminative method focuses on binary classification, often called CNN-C, which distinguishes an object from its background. The Generative method, often called CNN-M focuses on learning a robust similarity function, which aims to find a match to the object template from a specified region(Li et al. 2018).

There are a few common modern approaches to object detection, one being the use of CNN as described in the previous paragraph, Transformer based approaches, Vision Language models and lastly Hybrid models such as RetinaMask and EfficientDet(Sapkota et al. 2025) and lastly YOLO. YOLO is a model that has been trained on the COCO dataset to be able to detect sports balls. It is a small model and utilizes fast calculation speed.

2.3 Image Processing Algorithms

This section discusses how IPA can be used to extract relevant raw data for the PGMs. Image Processing Algorithm (IPA) is a term that can be used in equivalence to CV as in many cases IPA simulates human vision (Pitas 2000).

IPA can be broken down into 3 distinct categories, Low-Level, Intermediate level, and High-level vision algorithms. The low-level vision focuses on the most basic operations applied to raw pixels, the goal is to improve the image quality or extract very simple features. The intermediate level vision takes this cleaned low-level vision image and organizes these pixels into meaningful structures, this helps bridge the gap between pixel-level data and understanding of the objects in the image. Finally the high-level vision, is where Machine Learning (ML) comes into play and allows for object recognition (Pitas 2000).

2.3.1 Low-Level Vision

As stated before, low-level vision refers to the basic pixel-level processing that prepares raw images for higher-level interpretations (Ji 2013). In order to accomplish this various techniques can be applied, these enhance the image without altering underlying information within the image. The techniques that can be used are Edge Detection, this is where the operation identifies points on the image where brightness changes sharply, Filtering, which enhances certain aspects of an image, and finally Thresholding, which converts grayscaled images into binary images based on a threshold value (MetaEye 2025) by mapping intensity values above or below a threshold into distinct classes (such as the background and foreground). In practice this is commonly used in pre-processing video frames to enable more robust object detection and tracking.

2.3.2 Intermediate Level Vision

This level builds on the processing performed in low-level vision by assembling coherent objects and motion patterns. As discussed above this level bridges the gap between low-level and high-level visions (MetaEye 2025). Similar to Low-level vision, several techniques are commonly used at the intermediate level. These include segmentation, which divides the image into various segments or regions that share similar attributes (For example the watershed algorithms or k-means clustering), Object detection which identifies and localizes objects within each frame and finally Optical flow which estimates the motion between two consecutive frames (MetaEye 2025). Together all of these help to have a basic level of detection within the videos, such as tracking of a ball on the field. The final level being High-level vision is when classification of the objects come in, as this isn't in the scope of this project a further discussion is not needed.

2.3.3 OpenCV

OpenCV (Itseez 2024) is an Open Source Computer Vision library for image and video analysis and was developed by Intel over two decades ago. Originally written in C++/C but now has wrappers for Python, Java or Matlab (Culjak et al. 2012).

The library possesses various techniques for Low-level vision, intermediate level vision and High-level vision (Mohamad et al. 2015). Key functioning included are Threshold, Filtering, Finding edges, segmentation, background subtraction, and various other functions that can be applied to various different IPA (Marengoni and Stringhini 2011).

2.3.4 OpenCV Functions for Low- and Intermediate-Level Vision

This subsection provides a detailed breakdown of the low-level and intermediate-level vision functions available in OpenCV, which are relevant for image processing algorithms used in ball tracking. These functions are implemented in the library and can be accessed via Python bindings, as utilized in this project.

Low-Level Vision
As discussed previously low-level vision focuses on basic pixel-level processing that prepares videos or images for higher-level interpretations. The relevant functions available in OpenCV are summarized in Table 1.

Table 1: Low-Level Vision Functions from OpenCV

Operation	**Description**
Thresholding	Converts grayscaled images into binary images based on a threshold value, this is useful for separating the object from the background. Can use functions like `cv2.threshold()` (applies a fixed threshold to pixel intensities for binary segmentation) (Itseez 2023j) and `cv2.inRange()` (checks if the pixel values fall within a specified ranges for colour-based masking) (Itseez 2023k).

Operation	**Description**
Filtering	Enhances certain aspects of an image, such as smoothing or sharpening, which in turn helps improve ball visibility. Can use functions like `cv2.medianBlur()` (replaces each pixel with the median intensity of neighboring pixels to reduce noise) (Itseez 2023q) and `cv2.GaussianBlur()` (applies Gaussian smoothing, which blurs the image by averaging nearby pixels with weights that decrease with distance from the center, effectively reducing high-frequency noise while preserving edges better than uniform averaging) (Itseez 2023r).
Frame differencing	Computes the absolute difference between consecutive frames to detect motion. Can use `cv2.absdiff()` (calculates pixel-wise absolute difference for motion detection) (Szeliski 2010).
Finding edges	Identifies points where brightness changes sharply, which can help outline the ball against the court or field. Can use functions like `cv2.Canny()` (multi-stage edge detection which uses gradients and hysteresis thresholding) (Itseez 2023b) and `cv2.Sobel()` (computes image gradients for edge detection in horizontal or vertical directions) (Itseez 2023c).
Colour space conversion	Transforms images between different colour spaces, which can enhance colour-based segmentation of the ball. Can use `cv2.cvtColor()` (changes colour representation, for example going from BGR to HSV for better colour separation) (Itseez 2023d).

Operation	Description
Morphological operations	Techniques like dilation and erosion that refine binary images by removing noise and closing gaps in detected objects. Can use functions like `cv2.dilate()` (expands bright regions in binary images using a structuring element) (Itseez 2023l), `cv2.erode()` (shrinks bright regions to remove small noise) (Itseez 2023m), and `cv2.morphologyEx()` (applies advanced operations like opening or closing for shape refinement) (Itseez 2023n).

Intermediate-Level Vision
Building on low-level processing, intermediate-level vision assembles coherent objects and motion patterns. The relevant functions available in OpenCV are summarized in Table 2.

Table 2: Intermediate-Level Vision Functions from OpenCV

Operation	Description
Segmentation	Divides the image into segments or regions that share similar attributes, this helps isolate the ball. Can use functions like `cv2.findContours()` (detects contours in binary images for shape analysis) (Itseez 2023h), `cv2.connectedComponents()` (labels connected regions in binary images for segmentation) (Itseez 2023e), and `cv2.HoughCircles()` (detects circular shapes for ball-like object segmentation) (Itseez 2023i).

Operation	Description
Object detection	Identifies and localizes objects within each frame, this is crucial for tracking the ball. Can use shape-based filtering with `cv2.findContours()` (detects contours for shape analysis) (Itseez 2023h), `cv2.convexHull()` (computes convex hull for solidity) (Itseez 2023f), and `cv2.arcLength()` (calculates perimeter for circularity) (Itseez 2023g) to filter ball-like objects based on properties like aspect ratio, circularity, and solidity.
Background subtraction	Separates moving foreground objects from the static background, useful for isolating the ball in video frames. Can use `cv2.createBackgroundSubtractorMOG2()` (creates a MOG2 background subtractor for motion detection) (Itseez 2023a).
Optical flow	Estimates motion between two consecutive frames, useful for predicting ball movement. Can use functions like `cv2.calcOpticalFlowFarneback()` (dense optical flow estimation using polynomial expansion) (Itseez 2023o) and `cv2.calcOpticalFlowPyrLK()` (sparse optical flow using the Lucas-Kanade method) (Itseez 2023p).

2.4 Probabilistic Graphical Models

Probabilistic Graphical Models (PGMs) offer a powerful framework for representing and reasoning about probabilistic relationships, allowing computers to answer complex questions under uncertainty. They are widely used across various applications, including speech recognition, image segmentation, disease outbreak modeling, and CV, which is the primary focus of this project (Koller and Friedman 2009). PGMs combine probability theory and graph theory to create structured and interpretable models of complex distributions, and they scale effectively to high-dimensional problems. The structure and concepts throughout these following sub sections are based primarily on the foundational work presented by Koller and Friedman(Koller and Friedman 2009) as well as the courses presented by Daphne Koller on Coursera (Koller and Stanford 2025b; Koller and Stanford 2025a).

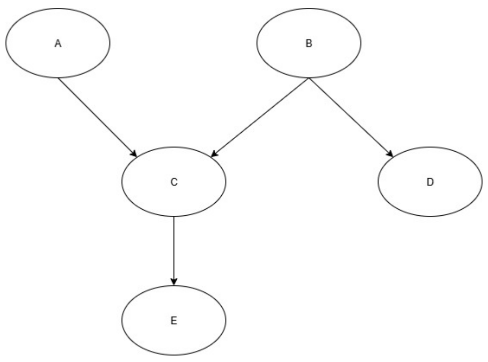
This section explores the key concepts of PGMs that are directly relevant to the object tracking approach developed in this project. It begins with a discussion on modeling, introducing the way graphical models are constructed and how probabilistic dependencies are encoded within them. Following that, the section discusses inference, describing how the various nodes in the model interact and how information is propagated throughout the graph to produce a coherent estimate. It then examines Gaussian factors, explaining how Gaussian distributions are used to represent uncertainty and how these distributions are applied to the outputs of image processing algorithms. Finally, it concludes by demonstrating how this probabilistic output is used for object tracking, and how it contributes to a more reliable and interpretable tracking system.

2.4.1 Modeling

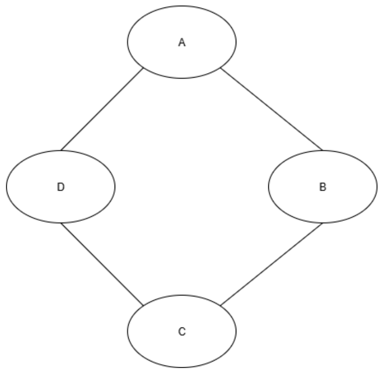
Models are declarative representations of our understanding of the world. They take in input data, apply structured reasoning, often through algorithms, and produce output that is relevant to the user's task. In the context of PGMs, the modeling is done through probabilistic reasoning, which is essential for managing uncertainty.

When creating a model in the context of PGMs, three main steps are typically followed. First, the Random Variables (RV) involved in the process are identified.

Next, a graphical model is constructed by selecting a representation suited to the problem. There are two main types of graphical models used in PGMs: Bayesian Networks (BNs), which are directed, and Markov Networks (MNs), which are undirected. Examples of both are shown in Figure 1.



(a) Example of Bayesian Network



(b) Example of Markov Network

Figure 1: Examples of graphical models

In Figure 1a, the BN is represented as a directed graph, where edges have arrows indicating conditional dependencies between random variables. A node with outgoing arrows is referred to as a parent, while nodes receiving those arrows are its children. For example, the arrow from node A to node C signifies that "C depends directly on A," or equivalently, "A influences C."

By contrast, the MN in Figure 1b is undirected, with edges lacking any direction. This reflects symmetric dependencies: neither node is assumed to influence the other in a causal sense. In this case, the undirected edge between A and B indicates that "A and B are directly dependent," without implying a specific direction of influence.

Finally, the last step is to define the factors required by the graph. For a BN, each factor corresponds to a Conditional Probability Distribution (CPD), a conditional probability ($P(A \mid B)$) distribution for a variable given its parents. If a node has no parents, its factor is simply a marginal distribution $P(A)$. For example, in the BN from Figure 1a, the associated distributions are:

$$P(A), P(B), P(C|A,B), P(D|B), P(E|C) \tag{1}$$

These are combined into one joint distribution ($P(X_1, \ldots, X_n)$) using the chain rule:

$$P(A,B,C,D,E) = P(A).P(B).P(C|A,B).P(D|B).P(E|C) \tag{2}$$

This joint distribution can then be used like any other multivariate distribution.

While this approach is manageable for small networks, it becomes computationally expensive for larger ones. This is where inference comes in. The next section discusses how inference methods, particularly belief propagation, enable efficient probabilistic queries without explicitly computing full joint distributions.

2.4.2 Inference

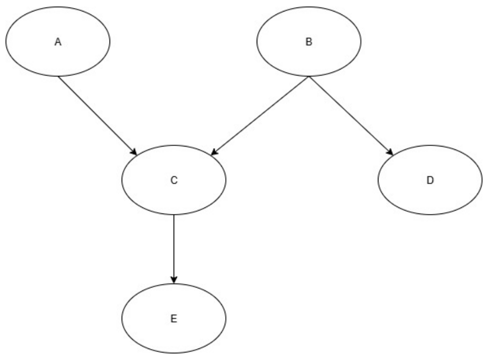
Inference is the process of calculating probabilities of RVs of interest given partial information. In practice, this means calculating the probability distribution of one RV conditioned on observed evidence from other RVs. This is very useful as in the real world not every RV is observed, therefore the use of inference provides reason for unobserved RVs.

All further explanations will be explained for BN because they can be encoded into a MN. This conversion from BN to MN involves removing the directional edges as well as making the inference more efficient. The conversion is achieved through moralization which is done in two steps:

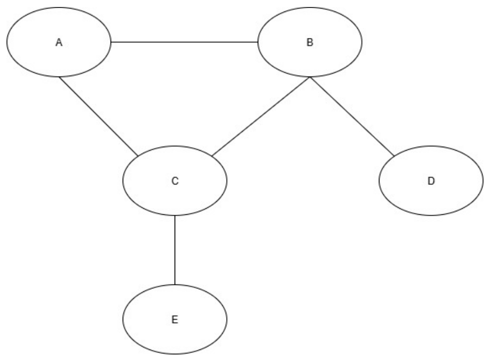
1) Connect all the parents of a common child - This means if two nodes have the same child an edge gets placed between them, also known as marrying the parents.

2) All directional edges are dropped and replaced with undirected edges.

This method is shown as an example in Figure 2, where it can be seen that the directed edges are converted to undirected edges and all parents who share children are joined.



(a) Example of Bayesian Network



(b) Example of Markov Network from a BN

Figure 2: Conversion of Bayesian Network to Markov Network

Now that all networks are MN, the focus shifts to the two main methods of applying inference, Belief Propagation (BP) and Belief Update (BU). These two methods are very similar, with a few differences.

BU is typically applied to a single variable or a small set of variables, using Bayes' Theorem directly:

$$\text{Posterior belief} = \frac{\text{Likelihood} \times \text{Prior belief}}{\text{Normalization factor}} \tag{3}$$

The purpose of BU is to refine the belief for a specific variable when new evidence arrives. This update is applied once per new observation, sequentially.

An example for this is if we looks at Figure 2b. If evidence at D is observed, Bayesian Updating can be applied to compute the belief for B first, as B is directly connected to D, then the new belief of B can be used to update the beliefs of A and C, which can be extended as far as needed.

In terms of mathematically explaining this, let evidence be observed at node D, denoted $D = d$. Using Bayesian Updating, we first update the belief for the neighboring node B:

$$P(B \mid D = d) = \frac{P(D = d \mid B)\,P(B)}{P(D = d)} \tag{4}$$

where the normalization factor is

$$P(D = d) = \sum_{B} P(D = d \mid B)\,P(B). \tag{5}$$

Next, we propagate the updated belief to the neighbors of B, namely A and C:

$$P(A \mid D = d) = \sum_B P(A \mid B) P(B \mid D = d), \tag{6}$$

$$P(C \mid D = d) = \sum_B P(C \mid B) P(B \mid D = d). \tag{7}$$

Finally, the belief at E can be updated via Σ over C:

$$P(E \mid D = d) = \sum_C P(E \mid C) P(C \mid D = d). \tag{8}$$

This demonstrates sequential Bayesian Updating: starting from the observed evidence and propagating through the network to refine the beliefs of all connected nodes.

The other method is BP, which is a message passing algorithm for inference in graphical models. This means that each node sends "messages" between each other about variables they have in common. The nodes then combine these messages into beliefs.

2.4.3 Gaussian Factors

For object tracking tasks such as in this project, Gaussian factors are especially useful as they can naturally represent noisy measurements of continuous variables, such as the ball's position across the video frames. Gaussian factors often refer to factors or potential functions defined by a multivariate Gaussian distribution over a set of RV. There are two main ways of describing Gaussian (Normal) distributions

1. Mean-Covariance form:

$$\mathcal{N}(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left\{ -\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu) \right\} \tag{9}$$

   where:

   - x: Vector of RVs
   - n: Length of x
   - Mean (μ): This is the "center" or average of the data.
   - Covariance (Σ): This represents how much the values vary and how the different dimensions move together. A higher covariance on the main diagonal represents a lower confidence and large variance from the mean.

2. Canonical (Information) form:

$$C(x;K,h,g) = \exp\left\{-\frac{1}{2}x^T Kx + h^T x + g\right\} \tag{10}$$

where:

- Information (Precision) Matrix K: Inverse of covariance, therefore a higher precision means lower uncertainty ($K = \Sigma^{-1}$)
- Information Vector h: Related to the mean ($h = K\mu$)
- Constant g: Adjusts the overall height but isn't always required.

The reason the canonical form is important for using Gaussian factors in a PGM is that it allows for more efficient inference. Inference for Gaussian factors in PGMs is performed via message passing on a cluster graph, following these steps:

1. Create Cluster graph: Group all the factors together into "Clusters" connected by shared variables.
2. Assign Factors to Clusters: Each cluster's "Potential" is a product of all the Factors assigned to it.
3. Send messages between Clusters: Each message marginalizes out the cluster's private variables, passing only the shared variables into its neighbor.
4. Compute Beliefs: Multiply incoming messages at each Cluster to get the final belief over its variables.

2.4.4 Hybrid Factors

Hybrid networks are PGMs that include both discrete and continuous RVs within the same modeling framework. This needs to be discussed because pure discrete networks aren't able to handle continuous measurements, and pure continuous networks can't represent discrete measurements. This section discusses what happens when a network or model has both discrete and continuous RVs.

Hybrid factors usually depicted mathematically as:

$$\phi(x,y) \tag{11}$$

where:

- x: Discrete RVs
- y: Continuous RVs

What this does is it handles the interaction between categorical decisions and real-valued outcomes. The joint distribution then factorizes as:

$$p(x,y) = \prod_{i=1}^{n} \phi_i(X_i, Y_i) \tag{12}$$

where each factor ϕ_i can be either purely discrete, purely continuous or truly hybrid. This factorization is important as it enables inference and represents dependencies.

Conditional Linear Gaussian (CLG) Networks

The most commonly used class of Hybrid Networks is a CLG network. A CLG network provides the structure to handle mixed discrete-continuous variables while keeping computational feasibility. A CLG network is generally preferred over general hybrid models as it is more tractable for inference, allowing for exact probabilistic computation rather than requiring approximation methods.

The CLG network is a hybrid network where:

- Discrete variables can have any parents. (Discrete or Continuous)
- Continuous Variables with discrete parents follow a Gaussian distribution whose parameters depend on the parents.
- Continuous Variables with continuous parents must follow a linear Gaussian model.

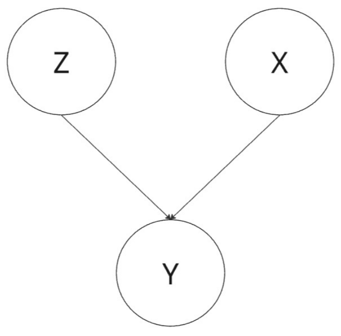


Figure 3: Example of a Conditional Linear Gaussian network showing discrete RV X and continuous RVs Y and Z

A general setup of a CLG network is shown in Figure 3, where Y is a continuous RV with discrete parent X and continuous parent Z, the conditional distribution is given by:

$$P(Y|Z, X = x) = N(Y; \mu_x(Z), \Sigma_x) \tag{13}$$

where:

- $\mu_x(Z) = a_x + b_x Z$ is a linear function of Z with parameters depending on the discrete state x.
- Σ_x is the covariance matrix also depending on x.

With all of this in mind, inference in CLG networks becomes more complex due to the mixture of discrete and continuous RVs. This is where Expectation Propagation (EP) allows for easier inference.

2.4.5 Expectation Propagation (EP)

EP is a deterministic approximation technique for BNs (T. P. Minka 2013). EP unifies two common approaches to inference techniques, the Kalman filter and Loopy BP, which is an extension of previously mentioned BP (T. P. Minka 2013). The method of EP is useful for Hybrid networks, as it approximates the belief states by retaining expectations such as means and variances (T. P. Minka 2013).

EP operates by approximating the posterior distribution through iterative moment matching between 'cavity' distributions (approximations excluding one factor's influence, denoted $q_{-i}(x)$) and 'tilted' distributions (cavity multiplied by the excluded factor, $q_{-i}(x) \cdot f_i(x)$). This process minimizes the Kullback-Leibler divergence (Thomas P Minka 2001; Heskes 2006).

EP uses an iterative approach which in many cases is computationally infeasible. A special case of EP is Assumed Density Filtering (ADF) or commonly called Moment Matching. This method is done by initializing all the approximating factors except the first one to be uniform and then only allowing a single pass (Bishop 2006, p. 510). This is necessary in the cases when data points arrive sequentially and the model must learn from each point and then discard the data before moving on to the next point (Bishop 2006, p. 510).

Key steps for the ADF algorithm are as follows:

1. **Initialization:** Start with simple approximations for all factors, assuming minimal prior knowledge.
2. **Sequential processing:** Calculate the 'cavity' and 'tilted' distributions and then match moments of the 'tilted' distribution to the Gaussian distribution, updating the approximation immediately.
3. **Single Pass:** Unlike the full EP's iterations, the ADF processes each data point once, discards it and then moves on to the next data point.

2.5 Gaps

Despite advancements in commercial ball tracking systems and computer vision techniques, significant gaps remain in accessible, real-time solutions for sports analytics. High costs and infrastructure requirements of systems like Hawk-Eye limit their use beyond professional leagues, while open-source alternatives often lack the precision needed for reliable ball trajectory estimation. Additionally, integrating computer vision outputs with probabilistic graphical models for inference in hybrid networks presents challenges, particularly in handling occlusions, fast motion, and sequential data processing. Approximation methods like Expectation Propagation offer potential, but their computational demands and accuracy trade-offs in dynamic environments require further optimization for low-cost, real-time applications.

3 Model Design

This section highlights the modular architecture, video footage, the role of PGMs, and the use of image processing algorithms. Figure 4 illustrates the overall pipeline of the system. The process begins with a sports video containing a ball. This video is passed through multiple image processing algorithms, each producing an independent estimate of the ball's position. These outputs are then fused using a PGM, which produces a final, unified estimate of the ball's position.

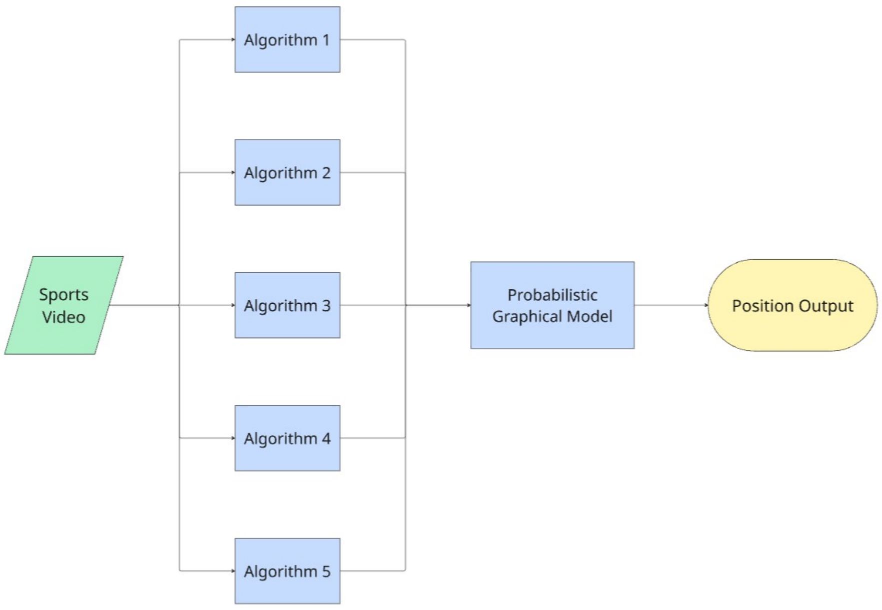


Figure 4: Overview of the model design

3.1 Video Footage Description

The sports video footage used in this project is sourced from a youtube video that displays a Court Level Tennis practice between Zverev and Djokovic (Apilado 2024). The video is 43 minutes long and downloaded in MP4 at 1440p60 quality. The footage is captured at court level, with the camera positioned behind one of the players, where the ball is mostly visible and clearly identifiable. The video will be cut down into shorter video clips where the ball is in play, this is done to ensure manageable processing time, and will be using 30 Frames Per Second (FPS). The video will be pre-processed to ensure consistent lighting and colour balance, which helps improve the performance of the image processing algorithms.

3.2 Image Processing Algorithm

This subsection builds on the discussion of IPA in the literature review (Section 2.3), where IPAs are categorized into various levels. The purpose of IPA in this project is to extract the raw data of the ball's position in each frame of the video and the reason for having multiple IPA is to ensure that if one algorithm fails to detect the ball in a frame, the other algorithms and the PGM can account for this failure and still provide reliable tracking of the ball. This subsection outlines the requirements needed of the IPA, a discussion and design of the various candidate algorithms, and the differences in the algorithms.

3.2.1 Requirements

To ensure that the IPAs effectively support ball tracking, they must meet specific requirements. These requirements are derived from the project's needs for real-time performance and robustness to dynamic sports environments, building on the challenges discussed in the literature review. The key requirements for the IPAs are outlined in Table 3.

Table 3: Requirements for Image Processing Algorithms

Requirement	**Description**
Robustness to noise and distractions	The algorithm must reliably track the ball without getting "distracted" by background noise, such as players, lines on the fields or courts, or spectators. This ensures consistent detection in cluttered sports footage.
Accuracy in ball center localization	The algorithm should precisely estimate the ball's center position, as well as minimizing errors that could affect trajectory prediction.
Process video footage at 30 FPS	The algorithm must process frames at 30 FPS to support close to real-time tracking, maintaining efficiency for video analysis without significant time delays.

Requirement	Description
Expand Scalability	The algorithm should handle varying video lengths, resolutions, and input formats with minimal performance degradation, allowing easy adaptation to diverse footage.

3.2.2 Algorithm Design

This subsection discusses the general design of the IPAs (Image Processing Algorithms), to ensure that the requirements in Section 3.2.1 are met. The design is presented as a Flow Diagram as per Figure 5, where the IPA needs to be able to take in sports video footage, process each frame, and output an estimated position. The per-frame pixel coordinate (x, y) is normalized by the frame width W and height H as follows:

$$x_{\text{norm}} = \frac{x}{W}, \qquad y_{\text{norm}} = \frac{y}{H} \tag{14}$$

which maps positions into the unit square $[0,1] \times [0,1]$. The idea behind using $[-1,-1]$ is that the later PGM will be able to identify easily that this input is not ideal and will ignore it since that specific algorithm didn't detect anything in that frame. These outputs then need to be outputted into a text file that can be used in the PGM, where the naming of the text file will be `alg_n.txt`, where $n \in \{1, ..., N\}$ and N is the number of algorithms used. Designing the general IPA to work in this way satisfies several of the reqirements, being that it can process each frame at 30 FPS, it can easily be scaled to various video lengths, and it can be adapted to different video formats. The rest of the requirements will be met according to the specific Low-Level and Intermediate-Level operations used in each algorithm, the position of which can be seen in Figure 5.

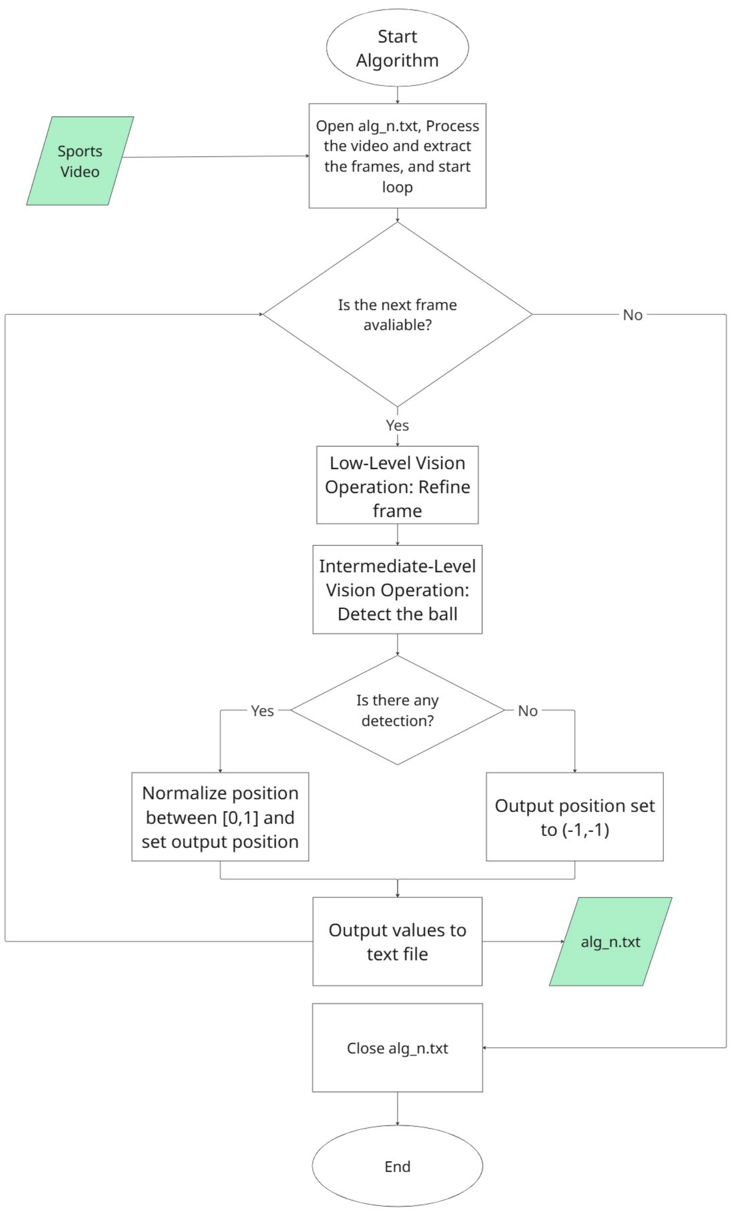

Start Algorithm
Sports Video
Open alg_n.txt, Process the video and extract the frames, and start loop
Is the next frame avaliable?
No
Yes
Low-Level Vision Operation: Refine frame
Intermediate-Level Vision Operation: Detect the ball
Is there any detection?
Yes
No
Normalize position between [0,1] and set output position
Output position set to (-1,-1)
Output values to text file
alg_n.txt
Close alg_n.txt
End

Figure 5: Flow diagram showing the flow of logic each of the IPA will follow

3.2.3 Candidate Algorithms

This subsection discusses and guides the design of the various candidate algorithms that can be considered for this project. The functions and operations of these algorithms are based on the OpenCV library (Itseez 2024), as discussed in Section 2.3.4. The aim of this subsection is to provide various combinations of operations that can be used to create different algorithms, which will then be tested and evaluated to determine the best performing algorithms for this project. The candidate algorithms are summarized in Table 4 and described further below the table.

Table 4: Candidate Image Processing Algorithms

Algorithm Name	**Low-Level Vision Operation Used**	**Intermediate-Level Vision Operation Used**
HSV Colour Thresholding	Median blur and morphological operations	Contour detection and shape analysis
Motion Tracker	Frame differencing, thresholding, and morphological operations	Contour detection, ellipse fitting, and shape analysis
Background Subtraction Tracker	Morphological operations	Background subtraction (MOG2), contour detection and shape analysis
Canny Edge and Hough Circles	Canny edge detection	Hough circle transform
Lucas-Kanade Optical Flow	N/A	Optical flow computation and feature tracking
Template Matching	Template correlation	Localization

The descriptions for each algorithm are as follows:

HSV Colour Thresholding: This algorithm uses colour-based segmentation by converting the frame to a HSV colour space. It applies median blur to reduce noise while preserving edges, followed by morphological operations like erosion and dilation to refine the binary mask. Contours are then detected, and shape analysis filters for circular shapes based on metrics like circularity and aspect ratio, making it effective for detecting the ball against varying backgrounds, however it may potentially be sensitive to lighting changes affecting colour perception.

Motion Tracker: This algorithm focuses on motion detection. It computes frame differencing to highlight moving objects, applies thresholding to create a binary image, and uses morphological operations to clean up noise. It detects contours and fits ellipses to candidate shapes, then applies shape analysis to filter based on elongation and size, which is particularly useful for tracking fast-moving balls but may struggle with slow or stationary objects.

Background Subtraction Tracker: This algorithm utilizes background modeling to separate foreground from background. It applies morphological operations to preprocess frames, uses MOG2 background subtraction to generate foreground masks, detects contours in the mask, and applies shape analysis to identify ball-like objects. It should be effective for static camera scenarios but may be affected by changing backgrounds or lighting variations.

Canny Edge and Hough Circles: This algorithm detects edges using the Canny edge detector to identify object boundaries. It then applies the Hough circle transform to find circular shapes. It should be particularly suited for detecting the ball when it's well-defined and circular, offering geometric precision but potentially struggling with deformed or partially occluded balls.

Lucas-Kanade Optical Flow: This algorithm implements sparse optical flow using the Lucas-Kanade method. It tracks feature points across frames to estimate motion trajectories. It should be ideal for following the ball's path over time, providing temporal continuity, but requires good feature initialization and may drift over long sequences.

Template Matching: This algorithm uses correlation-based template matching to locate the ball by comparing a reference template against the frame. It should be straightforward and effective when the ball appearance is consistent, but can be computationally intensive and sensitive to scale, rotation, or appearance changes.

These algorithms all have been designed and tested to ensure the algorithms meet the requirements outlined in Section 3.2.1. All of these algorithms use low-level vision to remove noise and make the algorithms more robust and free from distractions. The intermediate-level vision operations are utilized to accurately determine the center of the ball's position. Therefore based on Figure 5 and the descriptions above, these algorithms meet all of the requirements set.

3.3 Probabilistic Graphical Model

The purpose of incorporating PGMs in this project is to combine the outputs of multiple image processing algorithms in a principled and probabilistic manner. These algorithms, while effective individually, are not always reliable, they may occasionally fail to detect the ball or produce inaccurate estimations due to occlusion, motion blur, illumination variation or other visual disturbances.

A PGM allows for a robust fusion of these imperfect outputs by modeling both observed and latent variables, capturing the uncertainty inherent in each tracker's prediction. This subsection presents the design and construction of the PGM model used in the project, including its design requirements, structure, variable definitions, and how inference will be performed to estimate the ball's true position.

3.3.1 Design Requirements

In order to effectively design and implement the PGM for this project, a clear set of requirements must be defined. These requirements provide a foundation for understanding how the model should be structured, what functionality it must support, and how the final output should behave. In particular, the requirements address aspects such as uncertainty handling, output quality, and model scalability. Table 5 outlines the key requirements that the proposed PGM must satisfy.

Table 5: Requirements for the Probabilistic Graphical Model

Requirement	**Description**
Handle uncertainty	The PGM must account for noise or missing data in algorithm outputs by modeling them probabilistically.
Fuse multiple IPAs	Must combine several algorithm outputs at each frame into a multivariate normal distribution.
Temporal consistency	The model should account for the smooth, continuous nature of the ball's motion by modeling how its position evolves over time, ensuring that current estimates depend on previous positions.

Requirement	Description
Scalable inference	The model must support efficient inference algorithms that remain computationally feasible as the video length increases or as additional tracking algorithms are integrated.
Modular design	The structure should allow easy integration or removal of trackers without changing the core model.
Factor structure	The relationships between variables must be clearly defined using Gaussian Distributions.

3.3.2 Model Construction

A BN (Bayesian Network) was selected for this project as its strengths lie in its ability to explicitly model the dependencies between variables through the use of directed edges. This is BNs strength when compared with MN for this project, while both can represent complex dependencies and perform inference under uncertainty (Koller and Friedman 2009), the causal reasoning of the BN works well for this project as the image processing algorithms depend on the actual position of the ball.

Random Variables

There are 3 RVs per time step, these are presented in Table 6.

Table 6: Random Variables for the Probabilistic Graphical Model

RV	Symbol	Description	Domain
Ball position	$Pos_{x,y}$	Actual position of the ball on the screen in normalized coordinates (x and y values)	$[0,1] \times [0,1]$

RV	Symbol	Description	Domain
Algorithm outputs	Alg_n (for $n = 1, \ldots, N$)	Output from the n-th image processing algorithm	$[0,1] \times [0,1]$
Outlier Indicator	a_n (for $n = 1, \ldots, N$)	Discrete RV indicating if Alg_n is an outlier	$\{0 = \text{outlier}; 1 = \text{inlier}\}$

Since each image processing algorithm typically outputs a single (x, y) position estimate, this alone does not provide a continuous distribution over all possible ball locations. To meet the requirement of a multivariate normal distribution in the model, a Gaussian factor is applied to each algorithm's output. This is done by taking the algorithm's output as the mean μ_n of a multivariate Gaussian distribution and associating it with a covariance matrix Σ_n that captures the algorithm's expected positional uncertainty. The specific values for these covariance matrices, along with temporal motion parameters and outlier probability priors, will be determined through systematic evaluation in Chapter 5.

This transforms the single point estimate into a probabilistic relationship, centered around the algorithm's prediction with an associated covariance that represents its uncertainty. The resulting domain for each algorithm's output becomes $[0,1] \times [0,1]$, with a continuous distribution that reflects the inherent error or noise in the algorithm's estimate.

Sometimes the algorithms may produce irregular or wrong outputs, for example if the ball is occluded or there are distractions within the frame then one of the IPA may output a position that is far from the actual ball position. To account for this the PGM includes a discrete RV called a_n for each individual IPA. This RV indicates if the output is an outlier or inlier, this therefore produces a Hybrid network as there are now both discrete and continuous RVs.

Relationships

Now that the RVs have been defined, the next step is to define the relationships between them. It is evident that the output of each algorithm directly depends on the actual position of the ball as well as whether the output is an outlier. Therefore, there is a causal relationship between $Pos_{x,y}$ and Alg_n and between a_n and Alg_n, as illustrated in Figure 6. This implies that the algorithm outputs are conditionally dependent on both the true ball position and the Outlier Indicator.

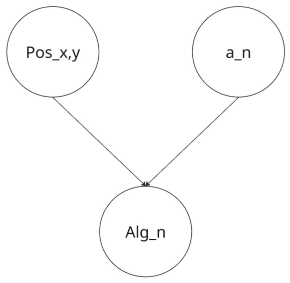


Figure 6: Relationship between $Pos_{x,y}$, Alg_n and a_n for single algorithm and time step

With this in mind, the full model can be defined as shown in Figure 7. This represents the final BN design used in this project for a single time step. (While the example illustrates five algorithms, the final number is still to be confirmed.)

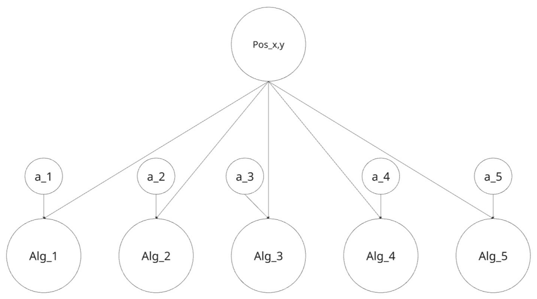


Figure 7: Bayesian Network showing the relationships between $Pos_{x,y}$ and the various algorithms used

Time-step connections

At this point in order for the ball to be tracked smoothly and efficiently the other frames need to be taken into account. This will be done with $Pos_{x,y}$ having a direct influence on the same RV in the next time step. From here the use of $Pos_{x,y}1$, $Pos_{x,y}2$, ..., $Pos_{x,y}n$ will be used per time step. Figure 8 shows the relationships between each time step. As can be seen, $Pos_{x,y}1$ has direct influence on $Pos_{x,y}2$, this means that the position of the ball in the next frame is dependent on the position of the ball in the current frame. This is important as it ensures that the ball's motion is modeled smoothly over time, reflecting its continuous trajectory across frames.

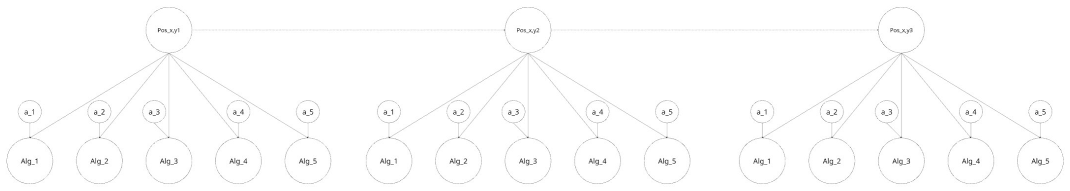


Figure 8: Bayesian Network showing the relationships between $Pos_{x,y}$, the various IPA used and the discrete Outlier Indicators at the different time steps

Factor Definitions

The probabilistic relationships in the BN are formalized through the following factor definitions:

- Temporal dynamics factor modeling the ball's motion between consecutive frames
$$P(Pos_{x,y}^{t}|Pos_{x,y}^{t-1}) \tag{15}$$
- Observation model factor relating algorithm outputs to true position and outlier status
$$P(Alg_n|Pos_{x,y},a_n) \tag{16}$$
- Outlier prior factor representing the probability that algorithm n produces an outlier
$$P(a_n) \tag{17}$$

These factors define the conditional dependencies within the hybrid network.

Mathematical Specification

These factor definitions now need to be formalized as Gaussian distributions. The mathematical specifications for the factors are as follows:

- Temporal dynamics factor:
$$P(Pos_{x,y}^{t}|Pos_{x,y}^{t-1}) = \mathcal{N}(Pos_{x,y}^{t}; Pos_{x,y}^{t-1}, \Sigma_{motion}) \tag{18}$$
- Observation model:
$$P(Alg_n|Pos_{x,y},a_n) = \begin{cases} \mathcal{N}(Alg_n; Pos_{x,y}, \Sigma_n) & \text{if } a_n = 1 \text{ [inlier]} \\ \mathcal{N}(Alg_n; c, \Sigma_o) & \text{if } a_n = 0 \text{ [outlier]} \end{cases} \tag{19}$$

  where:

  - c: A constant mean.
  - Σ_o: A large covariance matrix representing high uncertainty for outliers.

3.3.3 PGM Design

This subsection discusses the design flow of the PGM, as shown in Figure 9. The model should begin by reading the IPA outputs from their text files. For each time step (Frame), the model needs to set all of the initial means and covariance matrices. From there each the model should loop through each algorithm and perform EP to update the belief about the ball's position based on each algorithm's output and its outlier status. EP needs to be done here as this model is a Hybrid network and regular BP will not work for this model. The beliefs or messages are then fused from the last algorithm via Moment Matching as the inference method used is EP therefore there will be multiple difference guassians fusing messages together. Finally the outputs need to be converted to Mean-Covariance form and outputted to a text file that can be used for evaluation and to be overlayed with the original video footage to show the tracking.

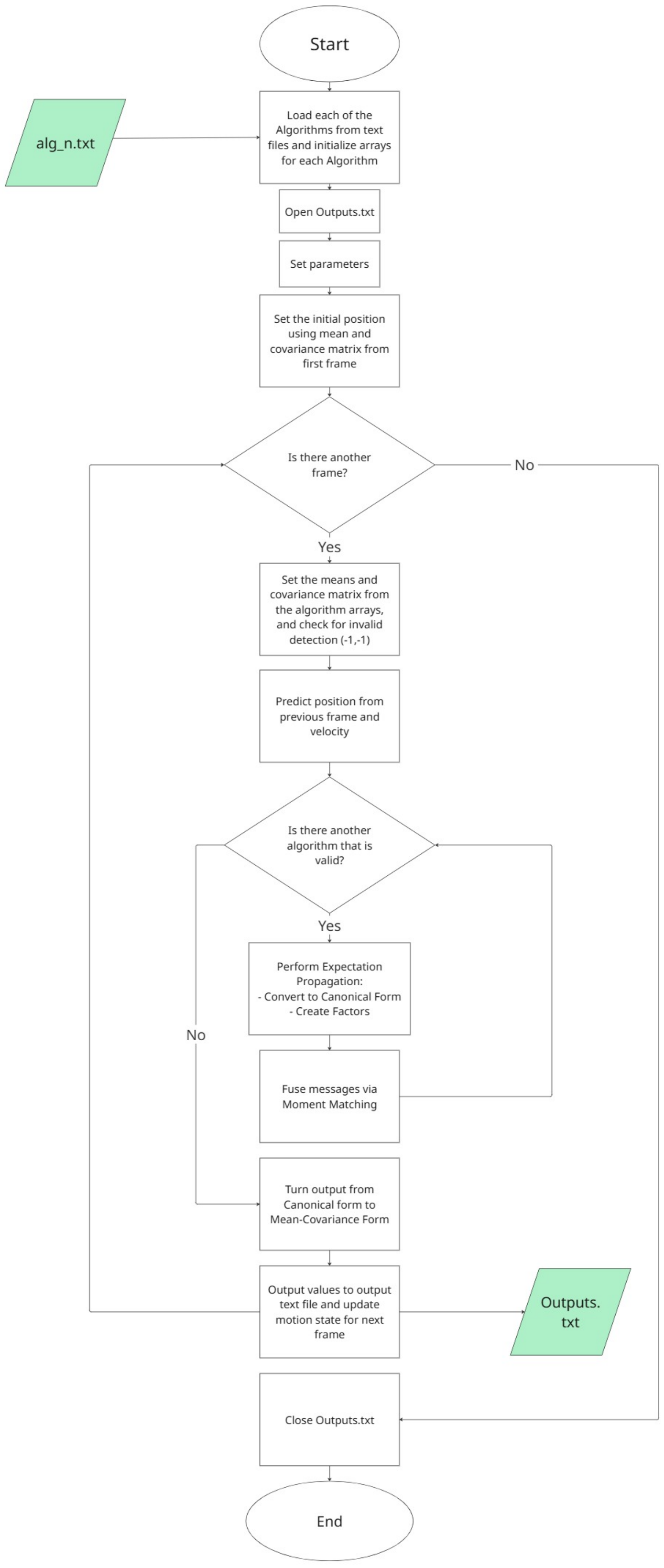

Start
alg_n.txt
Load each of the Algorithms from text files and initialize arrays for each Algorithm
Open Outputs.txt
Set parameters
Set the initial position using mean and covariance matrix from first frame
Is there another frame?
No
Yes
Set the means and covariance matrix from the algorithm arrays, and check for invalid detection (-1,-1)
Predict position from previous frame and velocity
Is there another algorithm that is valid?
Yes
No
Perform Expectation Propagation:
- Convert to Canonical Form
- Create Factors
Fuse messages via Moment Matching
Turn output from Canonical form to Mean-Covariance Form
Output values to output text file and update motion state for next frame
Outputs. txt
Close Outputs.txt
End

Figure 9: Flow diagram showing the flow of logic the PGM will follow

4 Implementation

This chapter discusses the implementation of the ball tracking system, focusing on IPA (Image Processing Algorithm), PGM (Probabilistic Graphical Model), and the way the two will communicate together and how data will flow through them. The chapter opens on the discussion of IPA implementation, followed by the implementation and foundations for how the PGM will be implemented. Finally, the chapter concludes with a discussion on how the system is integrated and how the data will flow throughout the system.

4.1 IPA Implementation

This subsection will detail the practical implementation of the IPA discussed and designed in Section 3.2. It will detail the IPAs implementation and Code architecture, parameter selection, output specification for each candidate algorithm and then final selection of algorithms. The implementation will ensure processing at 30 FPS while maintaining robustness to noise, distractions, or other factors which may affect tracking.

Based on the candidate algorithms discussed in Section 3.2.3, these IPAs will be implemented and tested. The final selection will be based on their strengths when compared with the other IPA, ensuring the selected algorithms complement each other in terms of ensuring the ball is tracked every frame by at least one algorithm and aren't too computationally heavy.

4.1.1 Implementation and Code Architecture

This subsection details the implementation of the IPAs described in Section 3.2.3. All implementations are developed in Python utilizing the OpenCV library (Itseez 2024) for computer vision operations.

Each IPA employs `cv2.VideoCapture` to process video footage frame-by-frame. Frames are analyzed according to the algorithm's design, with shared parameters including HSV bounds of $[27, 15, 150]$ to $[80, 255, 255]$, which are derived through iterative sampling of the ball's color across varying lighting conditions and the area constraints of $[20, 150]$ pixels, established via pixel counting at different distances to accommodate ball size variations. Pseudocode for these algorithms is provided in Appendix B.

If no ball is found in the algorithm, the output is set to $(-1, -1)$. Once the algorithm is done processing the outputs are written to a text file with naming convention of `alg_n.txt`, where n is the algorithm number. These algorithms will all be tested and evaluated in Chapter 5 to determine the best performing algorithms for this project. The final selection of algorithms will be confirmed in the Chapter 5.

HSV Colour Thresholding
The HSV Colour Thresholding algorithm forms the foundation of the system's low-level vision pipeline, leveraging colour segmentation for robust ball detection under varying illumination. Implemented in Python with OpenCV, it processes each frame sequentially, starting with noise reduction and progressing to shape-based filtering. The core architecture emphasizes modularity: colour conversion isolates the ball, morphological operations refine the mask, and contour analysis selects the most probable candidate based on geometric properties.

The processing pipeline begins with median blurring to preserve edges while suppressing noise, using `cv2.medianBlur`.

Colour space conversion to HSV enables threshold-based segmentation, invariant to brightness variations: `cv2.cvtColor` and `cv2.inRange`.

Morphological dilation and erosion refine the binary mask, filling gaps and removing distortions, using a 3x3 rectangular kernel: `cv2.dilate` and `cv2.erode`.

At the intermediate level, contour detection identifies potential ball regions, with selection criteria prioritizing circularity and solidity (area ratio to convex hull) to distinguish the ball from elongated or irregular shapes. A weighted scoring system ranks contours, selecting the top candidate and normalizing its centroid to $[0,1]$ for PGM integration.

Motion Tracker
The Motion Tracker algorithm detects and tracks the ball by combining motion detection with shape analysis, making it effective for fast-moving objects in dynamic scenes. Implemented in Python with OpenCV, it processes frames by first isolating motion through differencing, then refining with color cues and contour filtering.

The algorithm starts by converting frames to grayscale and HSV for dual processing: `cv2.cvtColor`.

Frame differencing highlights moving regions using absolute difference and thresholding: `cv2.absdiff` and `cv2.threshold`.

Finally the algorithm combines the masks via bitwise operations and isolates the ball:
`cv2.bitwise_and`.

Contour detection and shape analysis apply the same criteria as HSV Colour Thresholding (area, circularity, aspect ratio, solidity), prioritizing elongated shapes from motion streaks. The top-ranked contour's centroid is normalized to $[0,1]$ for PGM integration.

Background Subtraction Tracker
The Background Subtraction Tracker leverages background modeling to identify foreground motion, ideal for static camera setups. Using OpenCV's MOG2 algo-

rithm, it separates moving objects from the background, combining this with color filtering for robust ball isolation.

Initialization sets up the subtractor with tuned parameters: `cv2.createBackgroundSubtractorMOG2`.

Each frame is processed with HSV color masking and background application: `cv2.cvtColor`, `cv2.inRange`, and the subtractor's `apply` method.

Masks are combined and refined with morphological opening: `cv2.bitwise_and` and `cv2.morphologyEx`.

Contour detection uses the same selection criteria as HSV Colour Thresholding (area, circularity, aspect ratio, solidity), normalizing the best match to $[0, 1]$.

Canny Edge and Hough Circles

The Canny Edge and Hough Circles algorithm detects the ball through geometric shape recognition, focusing on circular edges for precise localization. Implemented in OpenCV, it preprocesses frames with color masking and blurring before applying edge detection and circle fitting.

Frame preprocessing involves HSV color masking and morphological cleanup: `cv2.cvtColor`, `cv2.inRange`, and `cv2.morphologyEx`.

Grayscale conversion and Gaussian blurring prepare for edge detection: `cv2.GaussianBlur`, and `cv2.bitwise_and`.

Canny edge detection identifies boundaries, followed by dilation for completeness: `cv2.Canny` and `cv2.dilate`.

The Hough Circle Transform detects circles with tuned parameters: `cv2.HoughCircles`.

The best circle's center is selected and normalized to $[0, 1]$ if detected.

Lucas-Kanade Optical Flow

The Lucas-Kanade Optical Flow algorithm tracks the ball by estimating feature point motion across frames, providing temporal continuity for dynamic trajectories. Using OpenCV's sparse optical flow, it initializes and updates points within color-masked regions.

Parameters are set for robust tracking using dictionaries for optical flow and feature detection.

Frames are preprocessed with color masking and morphology: `cv2.cvtColor`, `cv2.inRange`, and `cv2.morphologyEx`.

Feature detection and optical flow computation track points: `cv2.goodFeaturesToTrack` and `cv2.calcOpticalFlowPyrLK`.

Valid points within the mask are filtered, and their centroid is calculated and normalized to $[0, 1]$ if sufficient points are tracked.

Template Matching

The Template Matching algorithm locates the ball by correlating a reference template with each frame, offering straightforward detection when appearance is consistent. Using OpenCV's normalized cross-correlation, it preprocesses frames and applies a similarity threshold.

The template is loaded and prepared: `cv2.imread`.

Frames are masked and matched: `cv2.cvtColor`, `cv2.inRange`, `cv2.bitwise_and`, and `cv2.matchTemplate`.

The best match is located and thresholded: `cv2.minMaxLoc`.

The center is normalized to $[0, 1]$ if above threshold.

4.2 PGM Implementation

This section describes the practical realization of the PGM framework designed in Section 3.3. It addresses three principal challenges: representing Gaussian factors in a stable form suitable for message passing, performing EP on the hybrid BN, and handling the mixture model for probabilistic outlier rejection. The following subsections discuss the key technical decisions, algorithmic adaptations, and numerical considerations that enabled a robust and efficient implementation.

4.2.1 Software Framework and Architecture

The tracker was implemented in C++ to leverage computational efficiency and precision required for ball tracking processing. The implementation builds upon the EMDW framework developed at Stellenbosch University (Du Preez 2017), which provides foundational infrastructure for probabilistic graphical models. The Prlite library within EMDW supplies matrix and vector data structures (`RowMatrix`, `ColVector`), and the Eigen library provides linear algebra operations for matrix inversion and determinant computation.

This dual library approach was adopted because the Prlite matrices integrate well with EMDW's probabilistic modeling framework, while the Eigen library offers numerical algorithms for critical operations. Conversion functions bridge the two representations, transforming matrices between libraries only when necessary to minimize computational overhead.

4.2.2 Canonical Form Representation

The implementation employs Canonical (Information) form of Gaussian distributions rather than Mean-Covariance form. A Gaussian distribution in Canonical form is represented by its precision matrix and information vector. As shown in Equation 20 and 21 these are calculated using the mean μ and covariance Σ from the Mean-Covariance form:

$$\Lambda = \Sigma^{-1} \tag{20}$$

$$h = \Lambda\mu \tag{21}$$

This representation offers significant advantages for EP message passing. Product operations, which occur frequently during belief updates, reduce to simple addition in canonical form as shown in Equation 22.

$$\mathcal{N}(x;\mu_1,\Sigma_1) \times \mathcal{N}(x;\mu_2,\Sigma_2) \propto \mathcal{N}(x;\Lambda_1+\Lambda_2,h_1+h_2) \tag{22}$$

Similarly the division operations, required for cavity computation, become subtractions. This eliminates the need for repeated matrix inversions that would dominate the computational cost. The posterior distribution and all factor messages are maintained in canonical form throughout inference, with conversion to Mean-Covariance form performed only at the final output stage.

4.2.3 Expectation Propagation Inference

Standard BP algorithms are unsuitable for a hybrid network such as this one, being unable to effectively handle the mixture of discrete outlier indicators a_n coupled with continuous position variables. EP provides an approximate inference framework that marginalizes over discrete variables while maintaining tractable Gaussian beliefs over continuous variables.

The idea behind the next 2 subsections is that the EP tracker refines position estimates by: removing each algorithm's contribution (cavity), evaluating mixture components via Bayesian weighting, collapsing to a single Gaussian through moment matching, and updating messages in canonical form. This process automatically reduces the number of outliers through Mahalanobis distance based weighting while the canonical form ensures stability through efficient matrix operations. The result is a robust fusion method that adapts to varying measurement quality without manual tuning.

Cavity Distribution Computation

At each iteration for algorithm n, EP constructs a cavity distribution $q_{-n}(x)$ by removing the current factor's contribution from the posterior. In canonical form, this is computed as:

$$\Lambda_{-n} = \Lambda_q - \Lambda_n \tag{23}$$

$$h_{-n} = h_q - h_n \tag{24}$$

where Λ_q and h_q represent the current posterior, and Λ_n and h_n are the message parameters from algorithm n. The cavity distribution serves as the prior for evaluating the mixture model in the next step.

Message Update

After computing the tilted distribution (described later in this subsection under the Mixture-Based Outlier Model), the new message is obtained by division. Since the distributions are in canonical form, this division operation reduces to subtraction:

$$\Lambda_n^{\text{new}} = \Lambda_{\text{tilt}} - \Lambda_{-n} \tag{25}$$

$$h_n^{\text{new}} = h_{\text{tilt}} - h_{-n} \tag{26}$$

This update ensures the algorithm incorporates new information from each algorithm. The posterior is then updated by incorporating these new message:

$$\Lambda_q = \Lambda_{-n} + \Lambda_n \tag{27}$$

$$h_q = h_{-n} + h_n \tag{28}$$

This direct update approach ensures that each algorithm's contribution is fully reflected in the posterior after a single pass through all measurements, making the inference efficient for sequential processing.

Mixture-Based Outlier Model

Each algorithm's measurement likelihood is modeled as a two-component Gaussian mixture that marginalizes over the discrete outlier indicator a_n and weights $w_{\pi,a}^{(n)}$ representing the prior probability of each component. The mixture model is defined as follows:

$$p(Alg_n|Pos_{x,y}) = w_{\pi,\text{outlier}}^{(n)} \cdot \mathcal{N}(Alg_n; Pos_{x,y}, \Sigma_{\text{outlier}}) + w_{\pi,\text{inlier}}^{(n)} \cdot \mathcal{N}(Alg_n; Pos_{x,y}, \Sigma_n) \tag{29}$$

where:

- The outlier component (first term) has a large covariance $\Sigma_{\text{outlier}} = \sigma_0^2 I$ with $\sigma_0^2 = 1.0$, representing high uncertainty when the algorithm produces an invalid measurement. 1.0 is selected as it covers the entire normalized image space, effectively treating outliers as uniformly distributed.

- The inlier component (second term) has algorithm-specific covariance Σ_n, representing the regular covariance matrix of algorithm n under normal conditions.

Bayesian Component Weighting

An updated weight for each component is computed using Bayes' rule, incorporating both the prior mixture weights $w_{\pi,a}^{(n)}$ and the likelihood that this position is an outlier. This is done to determine how much each component should influence the final estimate.

The weights are computed as follows as per (Bishop 2006, pg. 438-439):

$$w_a \propto w_{\pi,a}^{(n)} \cdot \mathcal{N}(\mathrm{Alg}_n; \mu_{-n}, \Sigma_{-n} + \Sigma_a^{(n)}) \tag{30}$$

To ensure numerical stability, weights are computed in log-space also from (Bishop 2006, pg. 438-439):

$$\log w_a = \log w_{\pi,a}^{(n)} - \frac{1}{2} \log |\Sigma_{-n} + \Sigma_a^{(n)}| - \frac{1}{2} (\mathrm{Alg}_n - \mu_{-n})^T \left(\Sigma_{-n} + \Sigma_a^{(n)}\right)^{-1} (\mathrm{Alg}_n - \mu_{-n}) \tag{31}$$

The final term represents the squared Mahalanobis distance between the position and the cavity mean. With this the measurements close to the predicted position (small Mahalanobis distance) receive high inlier weights, while distant measurements are automatically classified as likely outliers.

Moment Matching for Mixture Collapse

Since EP requires a single Gaussian approximation, the two-component mixture is collapsed to a single Gaussian via standard moment matching (Thomas P Minka 2001), then converted to canonical form $(\Lambda_{\text{tilt}}, h_{\text{tilt}})$ for the message update.

4.2.4 Temporal Motion Modeling

To incorporate temporal information into the tracking framework, a constant velocity motion model is employed to predict the ball's position at time t based on its previous trajectory. The velocity vector v^{t-1} is estimated from the displacement between consecutive posterior estimates:

$$v^{t-1} = \mathrm{Pos}_{x,y}^{t-1} - \mathrm{Pos}_{x,y}^{t-2} \tag{32}$$

The prior mean for frame t is then predicted by extrapolating the previous position using the estimated velocity scaled by a damping factor β:

$$\mu_{\text{prior}}^{t} = \mathrm{Pos}_{x,y}^{t-1} + \beta v^{t-1} \tag{33}$$

where $\beta \in [0,1]$ controls the trust placed in the velocity prediction, with $\beta = 0$ representing a position-only model and $\beta = 1$ representing full constant velocity.

The optimal value of β will be determined in Chapter 5 through evaluation on test sequences. The uncertainty in this motion prediction is captured by the process noise covariance $\Sigma^t_{\text{prior}} = \sigma_p^2 I$, where σ_p^2 represents the variance in position due to unmodeled dynamics such as acceleration, deceleration, or changes in direction. This process noise allows the tracker to remain responsive to actual changes in the ball's motion while still benefiting from the smoothing effect of the velocity-based prediction.

For the first frame ($t = 0$), no velocity information is available, so an uninformative prior centered at the image center with high uncertainty is used instead.

4.2.5 Implementation Summary

The implementation meets the designed PGM requirements through a combination of canonical form representation, EP message passing, and mixture-based outlier modeling. The canonical form reduces computational cost by eliminating redundant matrix inversions, while the mixture model provides outlier rejection without manual threshold tuning. The complete system processes algorithm outputs frame-by-frame, producing posterior position estimates with associated uncertainty quantification suitable for downstream analysis and visualization.

4.3 System outputs and Data Flow

The system is configured as a pipeline that sequentially processes all IPAs, executes the PGM, and finally overlays the results onto the video file. The data flow is shown in Figure 4. The output works by taking the output of the PGM which will be the mean position and the covariance matrix. This is then overlayed onto the video footage as seen in Figure 10. Where the mean is red and the green circle is the standard deviation around the mean, as calculated from the covariance matrix. The pipeline is set up within the files in Ubuntu and is run through a shell script.

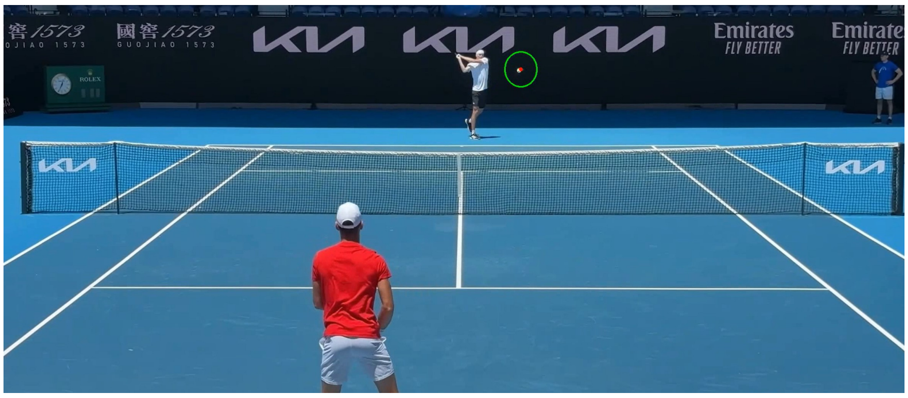


Figure 10: Example of output from the PGM overlayed with the original video footage

5 Evaluation

This chapter outlines the experimental setup for evaluating the performance of the ball tracking system, focusing predominantly on accuracy and computational efficiency. The chapter begins by describing the datasets and the various inputs required for the system to run, including the video footage and the ground truth data used for evaluation and to ensure reliable benchmarking. This is followed by a discussion of the evaluation metrics used to assess the system's performance, including the accuracy and success rate of the IPA, the accuracy of the PGM in keeping estimates within a standard deviation of the ground truth, and the overall computational efficiency measured by runtime. The experimental setup is then discussed, detailing the software environment used. This goes onto final experimentation and evaluation of the entire system, this is done to judge the systems robust nature and ability to handle different video footage. Finally, it concludes with the results and analysis of the system's performance based on the evaluation metrics discussed.

5.1 Datasets and Inputs

This section covers the video footage that is used as an input as well as how the ground truth data is generated for evaluation.

Video Footage

The video footage used in this project as described in Section 3.1 is sourced from a YouTube video (Apilado 2024), the video is a practice match between Novak Djokovic and Alexander Zverev. The footage is 1080p resolution at 30 FPS and are cut down to about 10 to 20 seconds long. The frame is 1080x1920 pixels and the camera is static throughout the footage. An example snippet of the footage is shown in Figure 11.

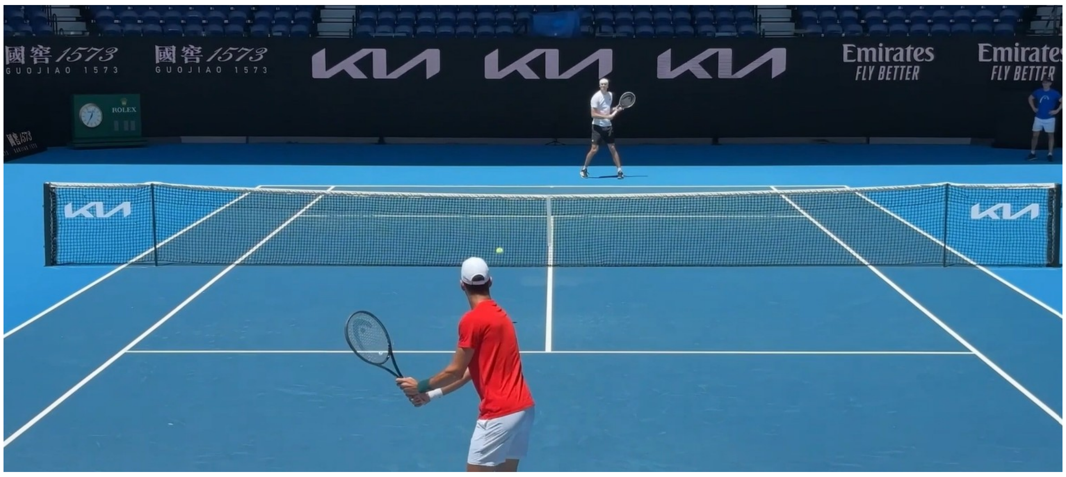


Figure 11: Example snippet of the video footage used in the project

The video shows clear lines on the court which is blue and the ball can be see relatively clearly as it is yellow against the background. The footage also shows a variety of ball trajectories, speeds, and occlusions, making it a suitable test case for the tracking system.

The evaluation of the IPA and PGM will be done using different footage to ensure that the system is robust and can handle various video footage. The ground truth data will be generated for each new footage used in the evaluation.

Ground Truth Data

The ground truth data for the video footage is generated through manual annotation. This is done in a basic Python script, using OpenCV, that allows the user to click on the ball in each frame of the video and then pressing the space bar to move to the next frame. The script also allows the user to skip frames if the ball is not visible, in which case the output for that frame is set to $(-1,-1)$. These recorded positions are then normalized and outputted to a text file called `groundtruth_`*n*`.txt`, the *n* indicating the specific video footage used. These values will be used to evaluate the performance of the system for both the IPA and the PGM.

5.2 Evaluation Metrics

The performance of all the various systems in the pipeline is discussed in this section. The metrics per system used are discussed in its subsection, that being the IPA and the PGM.

5.2.1 IPA Evaluation Metrics

The IPA performance is evaluated and selected using three main metrics: Mean Error, Success Rate, and Processing Time. This section also details how the Covariance Matrix, and the outlier weights $w_{\pi,a}^{(n)}$ for each IPA will be determined for the PGM.

Mean Error

The Mean Error is calculated as the average Euclidean distance between the predicted ball position and the ground truth position across all the frames where the ball is visible. This metric providdes a quantitative measure of the accuracy of each algorithm. The formula for Mean Error is given in Equation 34.

$$\text{Mean Error} = \frac{1}{N}\sum_{i=1}^{N}\sqrt{(x_i - x_{gt,i})^2 + (y_i - y_{gt,i})^2} \tag{34}$$

where:

- N is the number of frames where the ball is visible.

- (x_i, y_i) is the predicted position in frame i.
- $(x_{gt,i}, y_{gt,i})$ is the ground truth position in frame i.

Success Rate

The success rate of each of the IPA is calculated as the percentage of frames where the algorithm successfully detects the ball. Therefore a predicted position of (-1,-1) is considered a failure and any other position is considered a success. The formula for Success Rate is given in Equation 35.

$$\text{Success Rate} = \frac{\text{Number of Successful Detections}}{N} \times 100\% \tag{35}$$

Process Time

The processing time for each algorithm is measured as the total time taken to process an entire video footage. This metric is important to ensure that the algorithms can operate in real-time or near real-time conditions. The processing time is measured in seconds and is done entirely in code. This will be done by measuring the time taken to process Video 1 which is 17 seconds long.

Covariance Matrix and Weights Determination

The ground truths data will be compared with the outputs of each algorithm to determine an objective measure for determining the Covariance Matrix which will be used in the PGM. This will be done by calculating the standard deviation in the x and y positions and then taking the standard deviation, getting the variances using ($sd^2 = \sigma$) and using it to construct the covariance matrix, where σ_{xx} and σ_{yy} are the variances of the x and y positions, respectively. The covariance matrix Σ_n for algorithm n is defined in Equation 36. This will all be done on a Python script.

$$\Sigma_n = \begin{bmatrix} \sigma_{x,x} & \sigma_{x,y} \\ \sigma_{y,x} & \sigma_{y,y} \end{bmatrix} \tag{36}$$

The weights will be calculated based on the success rate of each IPA. The weight for the inlier component $w^{(n)}_{\pi,inlier}$ will be set to the amount of algorithm outputs within a 0.1 threshold of the ground truth position and that will be divided by the number of valid outputs the algorithm had. The weight for the outlier component $w^{(n)}_{\pi,outlier}$ will be set to $1 - w^{(n)}_{\pi,inlier}$. This ensures that algorithms with higher success rates have a greater influence on the final position estimate, while those with lower success rates are more likely to be treated as outliers.

5.2.2 PGM Evaluation Metrics

The PGM performance is evaluated using two main metrics: Standard Deviation Accuracy and Computational Efficiency. This subsection also highlights the parameter tuning process used to optimize the PGM performance, including the velocity damper β.

Standard Deviation Accuracy

The Standard Deviation accuracy metric evaluates how well the PGM maintains the estimated ball position within one standard deviation of the ground truth. This metric is calculated by taking the Euclidean distance between the mean of the PGM output and the ground truth positions. This distance is then compared with the standard deviation and if it falls within this range it is considered a success, these successes are all summed together and divided by the total number of frames to get the percentage of accuracy. The formula for Standard Deviation Accuracy is given in Equation 37.

$$\text{Std Dev Accuracy} = \frac{\text{Number of Frames within 1 Std Dev}}{N} \times 100\% \qquad (37)$$

Computational Efficiency

The computational efficiency of the entire system is measured by the runtime of the system after everything has been processed. This is done by measuring the time taken from start to end and is done entirely in code. The runtime is measured in seconds and the aim is to get the runtime to be as close to the actual length of the video footage as possible.

5.3 Experimental Setup and Results

The experimental setup for evaluating the system is conducted on Ubuntu 24.04.2 LTS using a computer equipped with a 12th Gen Intel Core i5-12450H processor and 32 GB of RAM. Python code is executed with Python 3.12.3, while C++ components are compiled using g++ 13.3.0. OpenCV library version 4.12.0 handles all computer vision tasks, and the EMDW framework supports PGM implementation as detailed in Section 4.2.1.

5.3.1 IPA Experimental Results

In order to evaluate the performance of each IPA, each algorithm has been run independently with three different video footage with lengths between 10 and 20 seconds long (Video 1 = 17 s, Video 2 = 17 s, Video 3 = 16 s). This is done to judge the Mean Error and Success Rate of each of the algorithms and get an average which can be used for tuning of the various parameters. The averages are presented in Table 7 and the full results are presented in Appendix C.

Table 7: IPA Performance Metrics

IPA	Mean Error	Success Rate (%)	Variance x	Variance y	Inlier weight	Process Time (s)
HSV Colour Thresholding	0.0423	73.89	0.0144	0.0107	0.8893	10.57
Motion Tracker	0.0164	80.21	0.0121	0.0141	0.9619	9.47
Background Subtraction Tracker	0.0462	72.79	0.0146	0.0045	0.8849	17.1
Canny Edge and Hough Circles	0.2094	95.52	0.0503	0.0395	0.5383	28.56
Lucas-Kanade Optical Flow	0.1830	91.67	0.0318	0.0713	0.5004	18.24
Template Matching	0.128	95.43	0.0261	0.0184	0.6506	22.04

Looking at the results in Table 7, the Motion Tracker has the lowest Mean Error with a moderate Success rate, however of those successes there is a high volume of inliers, meaning when the algorithm does detect the ball it is usually accurate. It also processed for the shortest amount of time making it the most computationally efficient. The next best performer was HSV Colour Thresholding, which had a slightly higher Mean Error but similar success rate and inlier rate. The processing time was also fairly similar to the Motion Tracker. The other IPA worth mentioning is Background Subtraction Tracker, because it had a similar Mean Error, Success Rate and Inlier Weight to HSV Colour Thresholding, however the processing time was significantly higher. The other three algorithms preformed very poorly, having high Mean Errors, and low inlier Weights. They did have a high number of successful detections, however judging by the Inlier weight most of these detections were inaccurate. The processing time for these three IPA are also significntly higher than what is desired. Judging by the tables in Appendix C, these IPA seem to fail in scenarios where there is more noise in the footage, this is because the first video has the least amount of noise and the last with the most.

Based on these results HSV Colour Thresholding, Motion Tracker and Background Subtraction Trackers have been selected to be used with the PGM as they have the best balance between accuracy, computational efficiency and inlier weights.

5.3.2 PGM Experimental Results

The PGMs have been evaluated using the same three videos as the IPA have been evaluated on. The metrics used for evaluation are the β, Accuracy (%) and the Efficiency. The β value have been evaluated in Table 15 in Appendix C and a graph in Figure 12 where the β value is changed from 0 to 1 in increments of 0.1 and the results are presented for each of the videos. Looking at the results, it is clear that a β value of 0.0 gives the best results across all three videos, with an average accuracy of 91.7 %.

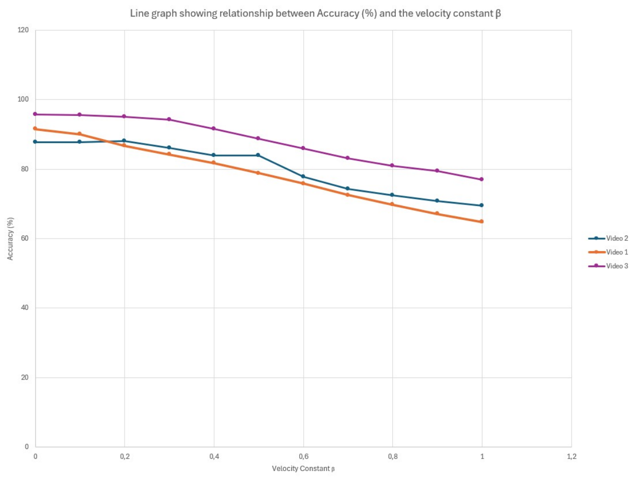


Figure 12: Graph showing the β value vs Accuracy (%) for all three videos

5.3.3 Final Experimentation of the Complete System

To validate the complete tracking pipeline with optimized parameters ($\beta = 0.0$, selected IPAs, and tuned covariance matrices), the system has been evaluated on 7 video clips, each between 10 and 20 seconds long. These clips feature diverse tracking scenarios including fast serves, slow rallies, and partial occlusions. The evaluation metrics include accuracy (%), mean error, and computational efficiency (runtime). Results averaged across all test clips demonstrate the system's robustness and are presented in Table 8.

Table 8: Final System Performance Metrics

Video number	Accuracy (%)	Mean Error	Video Length (s)	Runtime (s)
1	91.5	0.1605	17	18.4
2	87.8	0.0972	17	18.9
3	95.8	0.0718	16	20
4	77.1	0.3287	13	16.8
5	90.4	0.1952	12	13.9
6	90.7	0.17	17	19.5
7	95.9	0.0531	19	22.5
Average	89.89	0.154	15.86	18.57

The results in Table 8 indicate that the complete system achieves high accuracy across diverse scenarios, with an average accuracy of 89.89% and a mean error of 0.154. The runtime is relatively close to the video lengths, demonstrating near real-time performance, although this could be slightly improved with a better laptop. Notably, videos with higher occlusions show increased mean error and decreased accuracy, highlighting areas for potential improvement. The videos that performed the best overall had minimal occlusions and consistent lighting conditions. Overall, the system effectively integrates multiple tracking algorithms through the PGM, providing robust ball tracking in tennis footage.

5.4 Discussion of Results

This section provides an in-depth analysis of the experimental results obtained from evaluating the ball tracking system. It discusses the performance of individual IPA, the effect of parameter tuning on the PGM, and the overall system performance in various scenarios. Finally, it reflects on the limitations of the current approach and suggests potential avenues for future improvements.

5.4.1 Individual IPA Performance

Rounding out what has been discussed in Section 5.3.1 the Motion Tracker emerged as the most effective individual IPA, achieving the lowest mean error and a high inlier weight. This indicates that the Motion Tracker is adept at maintaining accurate ball position estimates. It had the best processing time and there go best computational efficiency. The other IPA performed considieriably worse than Motion Tracker, with higher mean errors and lower inlier weights. However HSV Colour Thresholding and Background Subtraction Tracker also had decent performances and out performed the other three IPA (Canny Edge and Hough Circles, Lucas-Kanade Optical Flow and Template Matching) by a significant margin. The poor performances of these three IPA can be attributed to their higher sensitivity to noise and occlusions in the video footagem which led to inaccurate detections. Even though these three IPA had high success rates, their high Mean Error and Inlier Weights indicate that most of these detections were inaccurate.

5.4.2 PGM Parameter Tuning Impact

The parameter tuning process for the PGM, being the tuning of the velocity damper β, had a significant impact on the system's performance. The results indicated that a β value of 0.0 yielded the best accuracy across all test videos. This suggests that relying solely on the current position estimates from the IPA, without incorporating velocity predictions, is more effective for ball tracking in this context. The likely reason for this is that tennis ball trajectories can be highly dynamic, with rapid changes in speed and direction that a constant velocity model may not accurately capture. By setting β to 0.0, the PGM effectively prioritizes the most recent observations from the IPA, allowing it to adapt quickly to sudden movements of the ball.

5.4.3 Overall System Performance

The complete ball tracking system demonstrated robust performance across a variety of scenarios. It achieves an average accuracy of 89.89% and a mean error of 0.154, this indicates that the system is effective at integrating multiple IPA outputs into the PGM. The performance was close to real-time, with the runtimes being above the video lengths with an average difference of 2.71 seconds. This means that the system is efficient enough for applications where the videos can be pre processed but not quite good enough for live tracking. The system performed particularly well in videos with minimal occlusions and consistent lighting conditions. However in scenarios with occlusions, the accuracy decreased and the mean error increased. This highlights the challenges posed by occlusions in ball tracking and suggests that further enhancements are needed to improve robustness in such conditions.

5.4.4 Limitations and Future Work

While the ball tracking system demonstrates strong performance with an average accuracy of 89.89%, several limitations present opportunities for future improvement.

Static Camera Assumption: The system requires static camera footage, limiting applicability where cameras follow the action. Extending to dynamic cameras would require camera motion compensation using feature-based tracking to stabilize frames before tracking.

Occlusion Handling: Performance drops significantly when the ball is blocked, with Video 4 achieving only 77.1% accuracy compared to 95.9% in Video 7 where the ball remained visible. The current PGM relies on direct IPA measurements which fail when the ball is hidden. While the system includes basic motion prediction, future work should incorporate more sophisticated trajectory models to better maintain tracking continuity during extended occlusions.

Real-Time Performance: Operating at 1.17× real-time speed (18.57s runtime for 15.86s videos) is insufficient for live video integration. While acceptable for post-processing, true real-time performance requires algorithmic optimization and GPU acceleration. The efficient Motion Tracker (9.47s) and HSV Colour Thresholding (10.57s) suggest reducing reliance on the slower Background Subtraction Tracker (17.1s) or implementing parallel processing.

Single Sport Limitation: Developed and tested exclusively on tennis footage with yellow balls on blue courts, the system requires adaptation of HSV color bounds and size constraints for other sports. Future work should explore generalization across sports with varying ball colors, sizes, and environments.

6 Conclusion

Ball tracking systems are an essential tool in modern sports analytics, officiating, and player performance evaluation. However, commercial systems such as Hawk-Eye and TRACAB remain inaccessible to most amateur clubs due to prohibitive costs and infrastructure requirements. This project investigated a cost-effective ball tracking approach that combines multiple Image Processing Algorithm through a Probabilistic Graphical Model to achieve robust tracking on consumer-grade hardware. The project performance is discussed in a techno-economic analysis in Appendix A.

The objectives set at the start of the project have been successfully achieved. The first objective was to develop and implement multiple IPA to estimate the ball's position from video footage. This was completed with six candidate algorithms evaluated, resulting in three being selected for the final system: HSV Colour Thresholding, Motion Tracker, and Background Subtraction Tracker. The second objective was to design a PGM-based fusion model to combine the outputs of individual algorithms, improving tracking accuracy and reducing errors from isolated failures. This was accomplished using Expectation Propagation inference, which achieved an average accuracy of 89.89% and a mean error of 0.154 across seven diverse test videos. The third objective was to evaluate and optimize the system to use the minimal number of algorithms required while maintaining acceptable performance. This was achieved by systematically comparing all six IPA and selecting only the three most effective ones. Finally, the fourth objective was to compare the model's performance against individual IPA. The results clearly demonstrate that the PGM-based fusion approach outperforms any single algorithm, with the combined system achieving superior accuracy and robustness, particularly in challenging scenarios with occlusions or varying lighting conditions.

In conclusion, this project was largely a success, achieving close to everything set out at the start. The system operates at 1.17× real-time speed on consumer-grade hardware, demonstrating practical feasibility for post-processing applications. While the system successfully demonstrates accurate ball tracking on consumer hardware, several areas require further development. Future work should focus on enhancing occlusion handling, achieving true real-time performance, and generalizing the system across different sports and camera setups, including dynamic camera footage. With these improvements, the proposed approach has the potential to democratize ball tracking technology, making it accessible to a wider range of sports organizations and enthusiasts.

References

Apilado, COURT LEVEL TENNIS - Liam (2024). *Djokovic and Zverev Full Court Level Practice | Points, Split-Screen, Groundstrokes etc. 2024 4K 60FPS.* Accessed: 2025-08-02. URL: `https://www.youtube.com/watch?v=n4TTjRjkB8I`.

Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning.* Accessed: 2025-09-29. Springer. ISBN: 978-0387310732.

Challa, Subhash (2011). *Fundamentals of Object Tracking.* Cambridge: Cambridge University Press. ISBN: 9780521192475.

Culjak, Ivana et al. (May 2012). "A brief introduction to OpenCV". In: *2012 Proceedings of the 35th International Convention MIPRO.* IEEE, pp. 1725–1730.

Daily News (2023). *Referee accuracy: VAR still debatable.* Accessed: 2025-07-15. URL: `https://dailynews.co.tz/referee-accuracy-var-still-debatable/`.

Dong, Xiaofei et al. (2014). "Infrared Dim and Small Target Detecting and Tracking Method Inspired by Human Visual System". In: *Infrared Physics & Technology* 62, pp. 100–109. DOI: `10.1016/j.infrared.2013.10.010`.

Du Preez, J. (2017). *EMDW Workspace.* Accessed: 2025-04-15. URL: `https://bitbucket.org/jadupreez/workspace/overview`.

Hawk-Eye Innovations (2024). *Our Technology.* Accessed: 2025-07-13. URL: `https://www.hawkeyeinnovations.com/technology`.

Heskes, Tom (2006). "Approximate inference: EP and loopy belief propagation". In: *Probabilistic graphical models*, pp. 205–210.

Itseez (2023a). *Background Subtraction.* Accessed: 2025-10-04. URL: `https://docs.opencv.org/4.x/d1/dc5/tutorial_background_subtraction.html`.

— (2023b). *Canny Edge Detection.* Accessed: 2025-10-04. URL: `https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html`.

— (2023c). *Canny Edge Detection.* Accessed: 2025-10-04. URL: `https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html`.

— (2023d). *Changing Colorspaces.* Accessed: 2025-10-04. URL: `https://docs.opencv.org/4.x/df/d9d/tutorial_py_colorspaces.html`.

— (2023e). *Connected Components.* Accessed: 2025-10-04. URL: `https://docs.opencv.org/4.x/d4/d73/tutorial_py_contours_begin.html`.

— (2023f). *Contour Features.* Accessed: 2025-10-04. URL: `https://docs.opencv.org/4.x/dd/d49/tutorial_py_contour_features.html`.

— (2023g). *Contour Features.* Accessed: 2025-10-04. URL: `https://docs.opencv.org/4.x/dd/d49/tutorial_py_contour_features.html`.

— (2023h). *Contours: Getting Started.* Accessed: 2025-10-04. URL: `https://docs.opencv.org/4.x/d4/d73/tutorial_py_contours_begin.html`.

— (2023i). *Hough Circle Transform.* Accessed: 2025-10-04. URL: `https://docs.opencv.org/4.x/da/d53/tutorial_py_houghcircles.html`.

— (2023j). *Image Thresholding.* Accessed: 2025-10-04. URL: `https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html`.

Itseez (2023k). *Image Thresholding*. Accessed: 2025-10-04. URL: https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html.

— (2023l). *Morphological Transformations*. Accessed: 2025-10-04. URL: https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html.

— (2023m). *Morphological Transformations*. Accessed: 2025-10-04. URL: https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html.

— (2023n). *Morphological Transformations*. Accessed: 2025-10-04. URL: https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html.

— (2023o). *Optical Flow*. Accessed: 2025-10-04. URL: https://docs.opencv.org/4.x/d4/dee/tutorial_optical_flow.html.

— (2023p). *Optical Flow*. Accessed: 2025-10-04. URL: https://docs.opencv.org/4.x/d4/dee/tutorial_optical_flow.html.

— (2023q). *Smoothing Images*. Accessed: 2025-10-04. URL: https://docs.opencv.org/4.x/dc/dd3/tutorial_py_filtering.html.

— (2023r). *Smoothing Images*. Accessed: 2025-10-04. URL: https://docs.opencv.org/4.x/dc/dd3/tutorial_py_filtering.html.

— (2024). *OpenCV: Open Source Computer Vision Library*. Version 4.12.0. Accessed: 2025-10-11. URL: https://opencv.org.

Ji, Qiang (2013). *Introduction to 3D Vision*. Accessed: 2025-09-29. URL: https://sites.ecse.rpi.edu/~qji/CV/3dvision_introx.pdf.

Kamble, Pravin R., Avinash G. Keskar, and Kishor M. Bhurchandi (2019). "Ball Tracking in Sports: A Survey". In: *Artificial Intelligence Review* 52.3, pp. 1655–1705. DOI: 10.1007/s10462-018-9629-5.

Koller, Daphne and Nir Friedman (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press. ISBN: 9780262013192.

Koller, Daphne and University Stanford (2025a). *Probabilistic Graphical Models 2: Inference*. Coursera. Accessed: 2025-09-29. URL: https://www.coursera.org/learn/probabilistic-graphical-models-2-inference/home/module/1.

— (2025b). *Probabilistic Graphical Models: Representation*. Coursera. Accessed: 2025-09-29. URL: https://www.coursera.org/learn/probabilistic-graphical-models/home/module/1.

Labayen, Mikel et al. (2014). "Accurate Ball Trajectory Tracking and 3D Visualization for Computer-Assisted Sports Broadcast". In: *Multimedia Tools and Applications* 73.3, pp. 1819–1842. DOI: 10.1007/s11042-013-1667-7.

Li, Peixia et al. (2018). "Deep Visual Tracking: Review and Experimental Comparison". In: *Pattern Recognition* 76, pp. 323–338. DOI: 10.1016/j.patcog.2017.11.024.

Marengoni, M. and D. Stringhini (Aug. 2011). "High level computer vision using OpenCV". In: *2011 24th SIBGRAPI Conference on Graphics, Patterns, and Images Tutorials*. IEEE, pp. 11–24.

MetaEye (2025). *Three Levels of Computer Vision*. Accessed: 2025-09-29. URL: https://www.metaeye.co.uk/three-levels-of-computer-vision?utm_source=chatgpt.com.

Minka, T. P. (2013). "Expectation propagation for approximate Bayesian inference". In: *arXiv preprint arXiv:1301.2294*. Accessed: 2025-09-29. URL: https://arxiv.org/abs/1301.2294.

Minka, Thomas P (2001). "Expectation propagation for approximate Bayesian inference". In: *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann, pp. 362–369.

Mirzaei, Behzad et al. (2023). "Small Object Detection and Tracking: A Comprehensive Review". In: *Sensors* 23.15. ISSN: 1424-8220. DOI: 10.3390/s23156887. URL: https://www.mdpi.com/1424-8220/23/15/6887.

Mohamad, Mohd et al. (2015). "A review on OpenCV". In: *Universitas Malaysia Terengganu* 3, p. 1.

Oliva-Lozano, José María et al. (2022). *Performance Tracking in Professional Football*. Accessed: 2025-07-15. URL: https://www.isspf.com/articles/performance-tracking-in-professional-football/.

Pal, Subhadip Kumar et al. (2021). "Deep Learning in Multi-object Detection and Tracking: State of the Art". In: *Applied Intelligence* 51.9, pp. 6400–6429. DOI: 10.1007/s10489-021-02283-1.

Pitas, Ioannis (2000). *Digital Image Processing Algorithms and Applications*. John Wiley & Sons. ISBN: 978-0471377391.

Premier League (2020). *How offsides are determined by VAR*. Accessed: 2025-07-15. URL: https://www.premierleague.com/en/news/1488423.

Sapkota, Rachana et al. (2025). *RF-DETR Object Detection vs YOLOv12: A Study of Transformer-based and CNN-based Architectures for Single-Class and Multi-Class Greenfruit Detection in Complex Orchard Environments Under Label Ambiguity*. arXiv preprint. arXiv: 2504.13099 [cs.CV]. URL: https://arxiv.org/abs/2504.13099.

Soleimanitaleb, Zahra, Mohammad Amin Keyvanrad, and Amir Jafari (Oct. 2019). "Object Tracking Methods: A Review". In: *2019 9th International Conference on Computer and Knowledge Engineering (ICCKE)*. IEEE, pp. 282–288. DOI: 10.1109/ICCKE48569.2019.8964846.

Stockman, George and Linda G. Shapiro (2001). *Computer Vision*. 1st. USA: Prentice Hall PTR. ISBN: 0130307963.

Szeliski, Richard (2010). *Computer vision: algorithms and applications*. Springer Science & Business Media.

TennisNerd (Aug. 2024). *Electronic Line Calling: Hawk-Eye Technology Explained*. Accessed 2025-07-15. URL: https://www.tennisnerd.net/articles/electronic-line-calling-hawk-eye-technology-explained/39915.

The Guardian (2025). *Rise of the machines: amid AI outrage, technology can be a force for good in sport*. Accessed: 2025-07-15. URL: https://www.theguardian.com/sport/2025/jul/15/rise-of-the-machines-ai-outrage-technology-tennis-sport.

Tracab (2024). *TRACAB: The Worlds Leading Sports Tracking System*. Accessed: 2025-07-13. URL: https://tracab.com.

Wiebe, Andrew (Apr. 2013). *MLS Commissioner Don Garber says league won't adopt goal-line technology by 2014*. Archived via the Wayback Machine; accessed 2025-07-15. URL: https://web.archive.org/web/20150919114824/http://www.mlssoccer.com/news/article/2013/04/25/mls-commissioner-don-garber-says-league-wont-adopt-goal-line-technology-2014.

Wong, Yu-Po (Ken) (2016). *Low-cost Tennis Line Call System with Four Webcams*. Tech. rep. Accessed 2025-07-15. Stanford, CA 94305: Department of Applied Physics, Stanford University. URL: https://web.stanford.edu/class/cs231a/prev_projects_2016/final_report_v2.pdf.

World Intellectual Property Organization (2023). *Goal-line technology – Getting it right*. Accessed 2025-07-15. URL: https://www.wipo.int/en/web/wipo-magazine/articles/goal-line-technology-getting-it-right-37357.

XbotGo (2024). *XbotGo Chameleon – AI-Powered Sports Tracking Camera*. Accessed 2025-07-15. URL: https://xbotgo.com/pages/xbotgo-chameleon.

Zhao, Yuwei et al. (Aug. 2015). "An Overview of Object Detection and Tracking". In: *2015 IEEE International Conference on Information and Automation (ICIA)*. IEEE, pp. 280–286. DOI: 10.1109/ICInfA.2015.7279340.

A Project Budget and Time

B IPA and PGM Pseudocode

C Experimental Results

Table 9: HSV Colour Thresholding Performance Metrics

Video number	Mean Error	Success Rate (%)	Variance x	Variance y	Inlier weight
1	0.02	68.94	0.0069	0.0126	1
2	0.09	79.01	0.0275	0.008	0.7
3	0.0169	73.71	0.0089	0.0115	0.9681

Table 10: Motion Tracker Performance Metrics

Video number	Mean Error	Success Rate (%)	Variance x	Variance y	Inlier weight
1	0.0072	75.00	0.0138	0.0216	0.9931
2	0.0362	81.37	0.0118	0.0048	0.9094
3	0.0059	84.26	0.0108	0.0158	0.9831

Table 11: Background Subtraction Tracker Performance Metrics

Video number	Mean Error	Success Rate (%)	Variance x	Variance y	Inlier weight
1	0.0136	65.71	0.0071	0.0216	1
2	0.0922	77.68	0.0239	0.0066	0.7162
3	0.0328	74.98	0.0129	0.0120	0.9385

Table 12: Canny Edge and Hough Circles Performance Metrics

Video number	Mean Error	Success Rate (%)	Variance x	Variance y	Inlier weight
1	0.1294	90.87	0.013	0.012	0.85
2	0.1513	96.98	0.1	0.08	0.456
3	0.3476	98.71	0.038	0.0265	0.3089

Table 13: Lucas-Kanade Optical Flow Performance Metrics

Video number	Mean Error	Success Rate (%)	Variance x	Variance y	Inlier weight
1	0.0829	78.83	0.0142	0.0685	0.9068
2	0.1617	98.16	0.0112	0.0011	0.3133
3	0.3045	98.01	0.0064	0.0017	0.2811

Table 14: Template Matching Performance Metrics

Video number	Mean Error	Success Rate (%)	Variance x	Variance y	Inlier weight
1	0.0095	89.23	0.0136	0.0239	0.9745
2	0.2371	98.36	0.0307	0.0111	0.4271
3	0.1375	98.71	0.0341	0.0205	0.5501

Table 15: PGM and β Performance Metrics

Video	β	Accuracy (%)	Efficiency (s)
1	0.0	91.5	25.7
1	0.1	90	27.9
1	0.2	86.7	28.1
1	0.3	84.2	27
1	0.4	81.7	27.3
1	0.5	78.8	29.4
1	0.6	75.8	27.2
1	0.7	72.5	27.7
1	0.8	69.8	27
1	0.9	67.1	27.2
1	1.0	64.8	27.2
2	0.0	87.8	25.7

Video	β	Accuracy (%)	Efficiency (s)
2	0.1	87.8	26.1
2	0.2	88	25.6
2	0.3	86.1	26.7
2	0.4	84	26.3
2	0.5	84	26
2	0.6	77.8	26.4
2	0.7	74.3	26.1
2	0.8	72.4	26
2	0.9	70.8	26.1
2	1.0	69.5	26.3
3	0.0	95.8	26
3	0.1	95.6	26.3
3	0.2	95	26

Video	β	Accuracy (%)	Efficiency (s)
3	0.3	94.2	25.9
3	0.4	91.6	26.9
3	0.5	88.8	26.1
3	0.6	85.9	26.1
3	0.7	83.1	25.5
3	0.8	80.9	29.3
3	0.9	79.5	28.8
3	1.0	76.9	27.3