University of Newcastle

Final Report

Tennis Trainer

Author:
Thomas Bailey
Student Number: 3254803

Supervisor:
Dr Colin Coates

June 22, 2020

A thesis submitted in partial fulfilment of the requirements for the degree of Bachelor of Engineering in Electrical Engineering at The University of Newcastle, Australia.

Abstract

Tennis is a sport played by many people throughout the world, with skill ranging from amateur players at local clubs to professionals at international competitions. To provide challenging and repeatable tennis practise, tennis training machines have become an important tool for tennis players of all skill levels to improve their game. To make tennis training machines better and more accessible, a relatively low cost prototype tennis training machine has been designed by Timothy King of the University of Newcastle. To enable repeatable tennis training drills to be performed, a closed loop feedback mechanism must be designed that can track the tennis balls launched from the machine and use the trajectory information to calibrate various launch parameters.

This report examines the current state of the art for tennis ball tracking systems, Hawk Eye, as well as relevant technical research and theory behind modern ball tracking systems. A low-cost design for a near real-time tennis ball tracking system using off the shelf consumer hardware is also presented.

The proposed design utilises two cameras, each with a dedicated single board computer for parallel processing of initial computer vision algorithms, such as background subtraction and object detection. The object candidates from each camera are then transmitted using TCP/IP communication over WIFI to a main computer, which also runs the main tennis trainer control program.

Stereo rectification, correspondence and triangulation are then performed on the 2D object points to find the 3D position, after which they are placed into an acyclic graph. A modified version of the Viterbi algorithm known as Shift Token Transfer is then used to construct tracklets within the graph and a solution to the single source shortest path problem is used to find the minimum cost path through the graph, which represents the most likely ball trajectory. Finally, this trajectory can be compared with the desired reference ball trajectory, which is calculated based on the fundamental physics of tennis ball flight.

This research indicates through simulation results that a full court error of 12cm and an mean error of 4cm is the minimum achievable error using the proposed system. A scale model of the prototype system was developed, which demonstrates that the system can achieve near real time performance in complex environments for a relatively low cost.

Acknowledgements

A number of people have been key to achieving the outcomes of this project through their continued support and encouragement, which I would like to recognise in this acknowledgement.

Firstly, my academic supervisor Colin Coates, whose advice, support and insights have helped keep the project on track and moving in the right direction. Despite the somewhat tangential nature of this project to his field of expertise, Colin was always available to listen and provide guidance when it was needed.

Additionally, I would like to thank Josh Hindmarsh, for his feedback and support on the direction of the project and his help with seeing the big picture. I would also like to thank Taylor Young for his assistance and feedback on designing the 3D printed camera mounts and Christian Murchie, for loaning me his router for use on the project.

Finally, this project serves as the capstone to my education at the University of Newcastle. I would like to express my sincere gratitude to all those who have helped me both in and out of the classroom over the past 4 years, your influence has shaped the course of my life going forward.

Contributions

The project described in this thesis is based in the discipline of Electrical/Electronics Engineering. The key contributions to this project are listed below, with a time allocation breakdown found in Appendix A:

- Conducted review of the literature surrounding tennis ball tracking using computer vision.
- Conducted review of the technical background behind stereo vision systems.
- Constructed a model of tennis ball physics from fundamental principles.
- Designed a low cost prototype ball tracking solution using off the shelf components.
- Combined existing computer vision algorithms with novel stereo matching techniques and graph theory to achieve near real-time ball detection and triangulation.
- Conducted experiments to empirically determine appropriate object detection and filtering methods.
- Implemented Python code to control and manage camera resources and image processing locally.
- Implemented Python code to communicate between cameras via TCP/IP for synchronisation of cameras and transfer of ball candidate data.
- Designed and implemented a 1:10 scale model for testing and validation of prototype solution.

Thomas Bailey	Date
Dr Colin Coates	Date

Contents

1 Introduction

1.1 Project Motivation

The game of tennis originated in the 12th and 13th century, with the modern game of tennis experiencing rapid growth in the late 1960s [1]. Today, the game is played by millions of people across the world, ranging from amateurs at local tennis clubs to elite players in high level professional competitions. One of the clear hurdles for practising tennis techniques is the requirement to have another player of comparable skill level to play against for realistic training drills. To solve this problem, the first tennis training machine was designed by René Lacoste in 1928 [2], with significant technological evolution occurring over the next 90 years.

The majority of tennis training machines on the market today utilise pre-programmed routines to deliver balls with various velocities, trajectories and spins [3]. However, these machines are often expensive (>$5000 AUD) and even the most advanced machines do not provide any feedback to the user about shot performance. Furthermore, as the routines are pre-programmed, there is no feedback mechanism to address changes in atmospheric conditions or wear on the balls or machines.

To make tennis training machines better and more accessible, a relatively low cost prototype tennis trainer has been developed by Timothy King [4], which can perform pre-programmed tennis drills specified by the user. The machine allows for horizontal oscillation of -10 to +10 degrees, launch elevation of 0 to 50 degrees, top and back spin from 0 to 2500 rpm and launch velocity adjustments from 40 to 110 km/h. However, this machine currently has only basic closed loop feedback mechanisms using ball pitch length and therefore leaves room for substantial improvement.

1.2 Project Overview and Scope

This project aims to develop a low-cost method for providing near real time feedback to a tennis trainer machine utilising various computer vision and image processing techniques. With enhanced mechanisms for providing feedback on essential ball trajectory characteristics, it is hoped that the tennis trainer can achieve sufficient accuracy such that it can be used for affordable, accessible and quality tennis training drills for players of all skill ranges.

Additionally, if adequate mechanisms for closed loop feedback to the tennis trainer are achieved, this project aims to refine methods used for both 2D and 3D ball tracking and

trajectory interpolation to be used as a more advanced training tool, providing shot to shot feedback to tennis players training on the machine using visualisation techniques.

1.3 Requirements

Conditions that must be satisfied by the design in order to meet the primary project objectives.

- Vision system must be robust to handle changes in court texture, colour and background
- Capable of tracking tennis balls at up to 110km/h
- Software must be robust and bug free
- Use off-the-shelf components where possible
- System must be capable of near real-time performance. i.e. Processing complete before next ball served (approximately 3 sec [4])

1.4 Assumptions

Factors that are considered to be true throughout the course of the project.

- Functional tennis training machine is available for testing vision system and integration
- Access to relevant supporting documentation and permission to modify tennis training machine source code
- Will be used in conditions that comply International Tennis Foundation (ITF) regulations
- Will not be used in the rain or at night

1.5 Deliverables

The final project outcomes to be presented at the conclusion of the project.

- Computer vision source code and associated hardware capable of being integrated with existing tennis trainer machine
- Sufficient evidence to demonstrate the accuracy and implementation of the computer vision system

1.6 Constraints

Restrictions placed on the project to be observed throughout development.

- Computer vision system and associated hardware will be low-cost, approximately $300-500 AUD
- Delivered on time, 26th of June 2020

1.7 Report Outline

- **Section 1. Introduction** *(This section)*
- **Section 2. Review of Technical Background**
  Review of the current state of the art, followed by relevant domain specific research in ball tracking and computer vision techniques. Technical analysis of stereo vision is also provided, as well as the physics of tennis ball flight.
- **Section 3. System Overview**
  A high level contextual overview of the proposed solution. Describes the hardware and software utilised, simulation methods and flow charts for critical programs.
- **Section 4. Stereo Calibration**
  Provides a detailed explanation of how stereo calibration is typically conducted, highlights the issues surrounding dynamic calibration and provides justification for the stereo calibration method using in this prototype.
- **Section 5. Image Capture**
  Explains how images are captured using the selected hardware and discusses challenges involving the desired frame rate and resolution. Also details the method used to synchronise both cameras while recording.
- **Section 6. Background Subtraction and Object Detection**
  Details the method implemented for background subtraction and object detection. Explains the challenges presented by image noise and lighting conditions, and presents an experiment conducted to compare object detection methods.
- **Section 7. Rectification, Stereo Correspondence and Triangulation**
  Describes the methods used for image rectification and presents a novel method for stereo matching based on object features. Presents the method used for triangulation of 3D points and the testing method used to validate the entire computer vision pipeline.

- **Section 8. Trajectory Analysis**
  Explains how points spread over time in 3D space were formed into tracklets and scored based on their spatial and temporal relationship. A method for finding the best path through the tracklets is also presented using graph theory and a solution to the single source shortest path problem.
- **Section 9. Small Scale Testing and Evaluation**
  A 1:10 scale model of the prototype solution is presented and results from small scale experiments are discussed.
- **Section 10. Conclusions and Extensions**
  Summarises the project outcomes and highlights any components which must be addressed for full scale implementation. Introduces additional areas for which this research may be used in the future.
- **Appendix A - Time Allocation Breakdown**
  Shows the time utilised to complete each component of the project.
- **Appendix B - Tennis Ball Physics Model**
  Mathematical derivation and simulation of the tennis ball motion model.
- **Appendix C - Picamera Architecture**
  A visual representation of the hardware pipeline from Picamera to Raspberry Pi.
- **Appendix D - Stereo Camera Court Coverage**
  Shows the proposed orientation of stereo cameras and the court area covered using the standard angle of view of the PiCam V2.

2 Review of Technical Background

2.1 State of the Art

The current state of the art for tennis ball tracking and visualisation systems is Hawk Eye, which utilises 10 cameras to provide near real time analysis of tennis matches worldwide with a mean error of 2.6mm [5]. This system is prohibitively expensive for the majority of tennis players and clubs however, with approximate installation costs exceeding $60000 USD [6].

The modern Hawk Eye system is an evolution from the original tracking system proposed by Owens in [7], using non-synchronised television cameras to triangulate ball positions during live tennis matches. To obtain high processing speeds, each camera was equipped with a dedicated computer for ball candidate segmentation and construction of 2D 'tracklets' on the image plane. 2D tracklets from multiple cameras were combined on another computer using the epipolar constraint as a noise filter to eliminate spurious tracks. A quadratic model was used to join tracklets at impact points, such as bounces and volleys, and initialise a Kalman filter to best estimate the impact point. Complete tracks were rendered in 3D using Microsoft's DirectX for visualisation within 5 seconds of the end of the rally [7].

2.2 Computer Vision Techniques

Background Subtraction and Object Detection

Due to the variability in lighting and increased processing required for colour based segmentation on 3D (RGB) arrays, an alternate approach must be considered for tennis ball tracking. Mao, Mould, and Subramanian [8] present several methods for high speed, robust background subtraction of tennis games using dynamic background models and domain knowledge of tennis games.

The single Gaussian image differencing method presented in [8] calculates a single Gaussian at each pixel in a grayscale image. Pixel intensities outside of two standard deviations from the mean are considered foreground pixels, with remaining pixels considered background pixels. Using the domain knowledge that rapidly moving tennis balls occupy an entirely new set of pixels each frame, image differencing is performed to extract only changes in the foreground of the image.

A learning rate is also included to control the rate at which the background image is updated each frame. Further filtering methods based on ball shape and size are used

to remove noise and extract likely ball candidates. It should be noted that stray tennis balls on the ground from previous shots are rejected using this method due to their low velocity, an advantage over other methods.

Indoor results show this method achieved 90.7% true positives and 1.02% false positives, with outdoor results differing substantially, with 39.5% true positives and 26.5% false positives. [8] shows this method is over twice as fast as other methods using clustering and circular Hough Transforms to detect tennis ball candidates [9]. However, the fastest implementation of the method only achieved a processed frame rate of 19.5fps on a 782x582 image, which is a significant limiting factor in real time applications.

Object Tracking

Correctly identifying, segmenting and tracking tennis balls is a challenging task, as the system is required to track a small high speed object that is subject to significant distortion during hits and bounces, warping due to camera rolling shutter, shutter speed, and changing lighting conditions as the ball moves around the court. An upper limit for the potential accuracy of a tennis ball tracking system was estimated to be 92% in [8], using 32400 manually labelled image frames from a live tennis match.

A ball tracking method is demonstrated in [6] that involves merging a series of frames by taking the maximum at each pixel of each frame. This merged motion frame shows a trace of the ball which is filtered to remove noise and retain only the data that represents a ball trajectory. However, it is not clear how this trajectory based selection is performed.

Other ball tracking methods using particle filters have been implemented [10]. However, the implementation of these methods is more complex than other ball tracking methods, which would significantly increase the computational costs on a real time system.

Zhou, Xie, Huang, *et al.* [11] propose a method for ball tracking using an acyclic graph based scheme, turning the ball tracking problem into a min-cost network flow problem. Once ball candidates are detected, a modified version of the Viterbi algorithm called shift token transfer is used to form tracklets within several overlapping time windows.

Tokens are initialised on the first ball candidates in the window, with the score of each node calculated with reference to a constant acceleration ball motion model. Tracklets from each window are merged using several rules regarding their temporal and spatial overlap, resulting in a continuous track, see Figure 1. This method is a sub-optimal approach [11], but has low computational complexity, is able to track balls through periods of occlusion and outperforms the current state of the art approach to ball tracking.

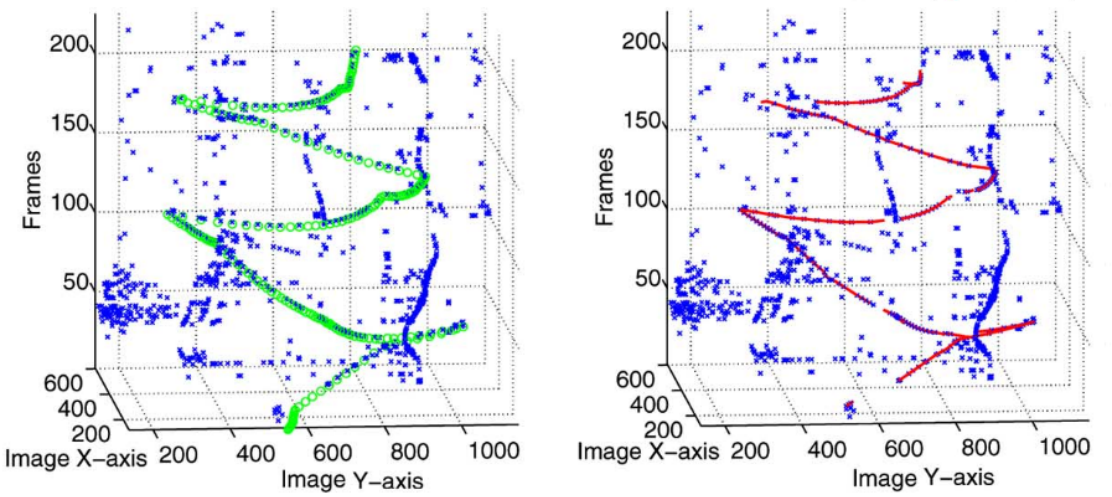

Figure 1: Ball tracking results from [11] presented in an acyclic graph. Blue x-marks represent ball candidates, green circles represent the ground truth and red lines indicate the final trajectory estimate

2.3 Camera Calibration and Localisation

In the pinhole camera model, light rays from an object pass through a common focal point before being projected onto the camera's image sensor or imager, this projected image is inverted with respect to the object. To simplify the mathematics, the focal point and image plane are switched, resulting in a new model where the light rays from the object are projected onto the image plane in the correct orientation [12], see Figure 2.

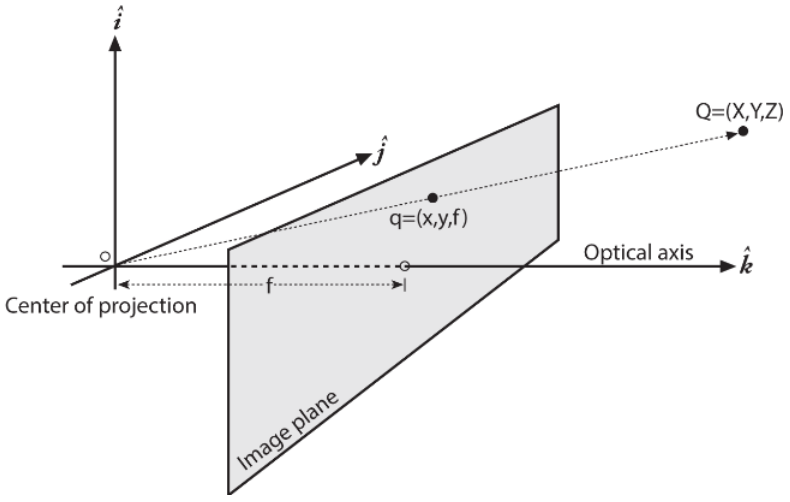

Figure 2: Modified Pinhole Camera Model [12]

This new model states that a ray from a point on the object ($\vec{Q}$) is projected onto the image plane at point ($\vec{q}$) and passes through a common point known as the centre of projection. The image plane is located between the object and the centre of projection at a distance f and is perpendicular to the optical axis. Two offset parameters (c_x & c_y) are introduced to correct for any displacement of the imager. Scaling parameters (f_x & f_y) are also introduced, as pixels on low cost imagers are often rectangular rather than square.

A projective transform is used to relate point $\vec{Q}$ in world space to point $\vec{q}$ in projective space using the 3x3 camera matrix (M), summarised by Eq 1. In this equation, $\vec{q}$ is represented in homogeneous coordinates, where the actual pixel coordinates (x, y) can be obtained by dividing through by w. Using homogeneous coordinates in projective space allows for all equivalent points to be represented by proportional values and also allows the construction of (M).

$$\vec{q} = M\vec{Q} \tag{1}$$

$$where: \qquad \vec{q} = \begin{bmatrix} x \\ y \\ w \end{bmatrix} \qquad M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \qquad \vec{Q} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Tangential and radial lens distortion are also factors that need to be included when calibrating the camera. Tangential distortion is introduced during manufacturing, where the imager is not parallel to the imaging plane, whereas radial distortion is due to both misalignment between the lens and imager and also from the use of spherical lenses over parabolic lenses. Radial distortion is corrected using the Taylor series expansion around the centre of the lens, where there is zero distortion, see Eq 2. Similarly, tangential distortion is corrected using Eq 3 & 4 [12].

$$x, y_{corrected} = x, y \times (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{2}$$

$$x_{corrected} = x + [2p_1 xy + p_2(r^2 + 2x^2)] \tag{3}$$

$$y_{corrected} = y + [p_1(r^2 + 2y^2) + 2p_2 xy] \tag{4}$$

Extrinsic camera parameters such as the rotation and translation of the object must also be taken into account during calibration. The rotational relationship is represented by a rotation matrix around each axis, combined into a single rotation matrix R. Translation

is represented using the translation vector $\vec{T}$ and effectively shifts the origin from the object space to the camera space, $\vec{T} = Origin_{obj} - Origin_{cam}$. The relationship between a point in object space $\vec{P_o}$ and a point in camera space $\vec{P_c}$ is shown in Eq 5.

$$\vec{P_c} = R(\vec{P_o} - \vec{T}) \tag{5}$$

Combining Eq 1 & 5 gives a system of equations that can be solved to obtain the intrinsic camera parameters. A planar calibration object is used to obtain a set of points at known locations, such as a black and white chessboard, see Figure 3.

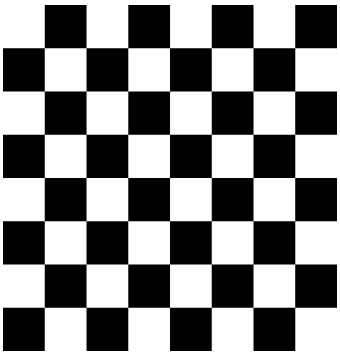

Figure 3: Chessboard Calibration Image

It is important to note that intrinsic camera parameters stay the same for different camera views, where the extrinsic camera parameters change depending on the object location. Thus, there are a total of 10 unknowns for each view, with 6 changing in every view. 8 equations are then obtained from the projective mapping of the four chessboard corner points to the 4 points on the image plane, given that each point has both an x and y coordinate. For high quality results, at least 10 unique chessboard images are required to solve the equations, due to the sensitivity of the the intrinsic parameters to noise [12].

The results for calibration without distortion are then used to solve the distortion parameters using Eq 2 - 4. The methods used to solve this set of equations is beyond the scope of this project, however open source computer vision libraries such as OpenCV provide fast, robust fitting algorithms that can be used to achieve this.

2.4 Epipolar Geometry and Stereo Imaging

Stereo imaging is the process by which two different simultaneous views of the same object can be used to triangulate the object's position in 3D space using epipolar geometry. There are 5 key steps required to obtain a 3D reprojection from two images.

- Undistortion: Mathematically remove radial and tangential lens distortion, as seen in Section 2.3.
- Stereo Calibration: Compute the spatial relationship between the two cameras.
- Rectification: Transform images to adjust for spatial relationship between cameras.
- Correspondence: Match features between the left and right images.
- Triangulation: Project 2D points to 3D using the feature disparity.

Epipolar Geometry

Epipolar geometry effectively uses two modified pinhole camera models, as described in Section 2.3. The epipolar plane is defined by the point in 3D space (P) and the centre of projection for both cameras (O_l & O_r), see Figure 4. The epipolar lines are the lines along the image plane where it is intersected by the epipolar plane. The epipolar constraint imposed by the geometry is a powerful tool, as any feature in one image must lie along the corresponding epipolar line in the other image. This greatly reduces the search space when matching features during stereo correspondence. The epipolar constraint also acts as a noise filter, as any noise present in one image will not correspond to points in the other image.

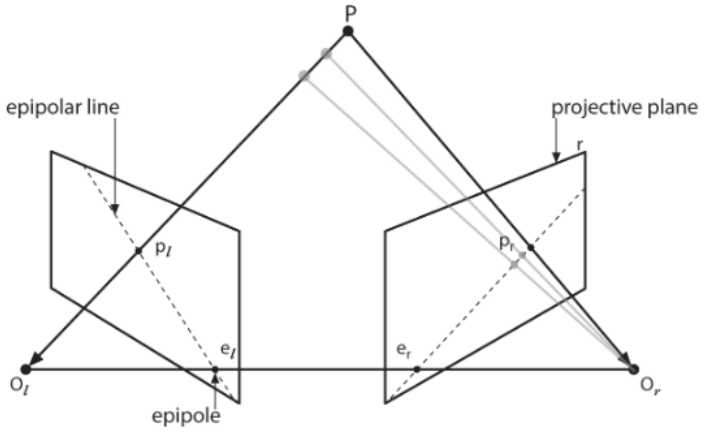

Figure 4: Epipolar Geometry [12]

The essential matrix (E) and the fundamental matrix (F) are used to relate the two images based on the physical relationship of the two cameras. The essential matrix contains the rotation matrix (R) and the translation vector ($\vec{T}$), where the fundamental matrix contains the same information, as well as the camera intrinsic matrix (M). F is therefore able to relate the two images in pixel coordinates, providing the cameras have already been calibrated independently.

Stereo Calibration

Stereo calibration aims to obtain the Rotation matrix (R) and translation vector ($\vec{T}$) which describe the spatial relationship between the two cameras. R is used to rotate the right image plane, making it coplanar to the left image plane, where $\vec{T}$ is used for triangulation. To obtain the spatial relationship between the two cameras, a method similar to that discussed in Section 2.3 is typically used, where images of a planar calibration object are captured from both cameras simultaneously and the points found in both images are used to solve for R and $\vec{T}$ using Eq 6 & 7. Where R_l & T_l are the rotation matrices and translation vectors from the left camera to the 3D point and from the right camera to the 3D point in the case of R_r & T_r.

$$R = R_r \cdot R_l^T \tag{6}$$

$$\vec{T} = \vec{T_r} - R \cdot \vec{T_l} \tag{7}$$

The method used to solve these equations is beyond the scope of this project, though open source stereo calibration functions found in OpenCV can be used to solve these equations for both camera views independently. These functions then use a robust Levenberg-Marquart iterative algorithm for find the minimum reprojection error and final solution for R and $\vec{T}$. As discussed in Section 2.3, typically 10 or more unique images are required to obtain a sufficiently accurate calibration with reprojection error less than 0.5. Reprojection error is a measure of calibration accuracy and is the mean distance between a detected keypoint in the pattern and a corresponding world point projected into the image.

Stereo Rectification

Stereo rectification is the process of row aligning both image planes in order to reduce the search space during correspondence matching and make the stereo correspondence problem less computationally intensive. This transformation is achieved by calculating the left and right rectification matrices (R_l & R_r), which transform the image coordinates of each camera such that they are coplanar and their principle rays are parallel to the vector sum of the way their original principle rays were pointing.

OpenCV uses Bouguet's Algorithm [12] to calculate R_l & R_r. Firstly, R is split between the two cameras, minimising reprojection distortion and placing the cameras in coplanar alignment. As the principle rays of each camera are now parallel, finding a vector $\vec{e_3}$ that is orthogonal to the two epipoles ($\vec{e_1}$ & $\vec{e_2}$) will produce a rectification matrix (R_{rect}) that takes the corresponding epipole to infinity with the epipolar line horizontal, see Eq 8.

$$R_{rect} = \begin{bmatrix} \vec{e_1}^T \\ \vec{e_2}^T \\ \vec{e_3}^T \end{bmatrix} \tag{8}$$

Stereo Correspondence

Stereo correspondence is the process of matching features in one image to the features in the other image, in order to calculate the disparity (d), which is the pixel difference in the x axis between the two image planes. This feature matching relies on the image planes first being correctly row aligned using stereo rectification, which reduces the feature search space to a single line of pixels in the other image, see Figure 5.

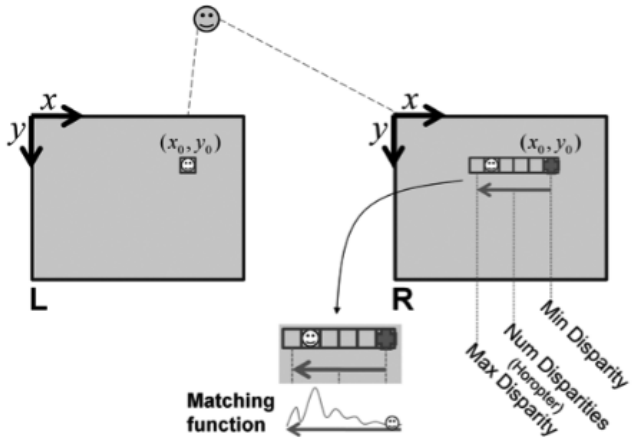

Figure 5: Row Aligned Rectified Images with Pixel Disparity [12]

Typically, feature matching is done using a technique called 'block matching' [12], which identifies highly textured areas in one image, matches them with corresponding areas in the other image then generates a disparity map for the entire image. Depending on the complexity of the image, this can be rather computationally intensive, especially if the image is relatively high resolution and the scene contains many highly textured areas.

In the context of this project, background subtraction and object detection have already been performed on each image independently, therefore it is only those objects that need to be matched between images, greatly reducing the computation required. A high level overview of a basic subtraction stereo matching algorithm is presented in [13], however a detailed explanation of how matching is achieved is not presented. The stereo matching algorithm designed for this project is presented in detail in Section 7.

2.5 Tennis Ball Physics

The coordinate system used for analysing tennis ball physics is shown in Figure 6, where ϕ is the launch elevation from the xy plane and θ is the launch azimuth from the y axis. Three forces act on the ball while in flight, the drag force F_d, the gravitational force F_g and the lift force due to the Magnus effect F_L. F_d is such that it always acts in the opposite direction to the velocity vector of the ball $\vec{v}$, where F_L acts perpendicular to v, with the direction determined by the direction of spin ω.

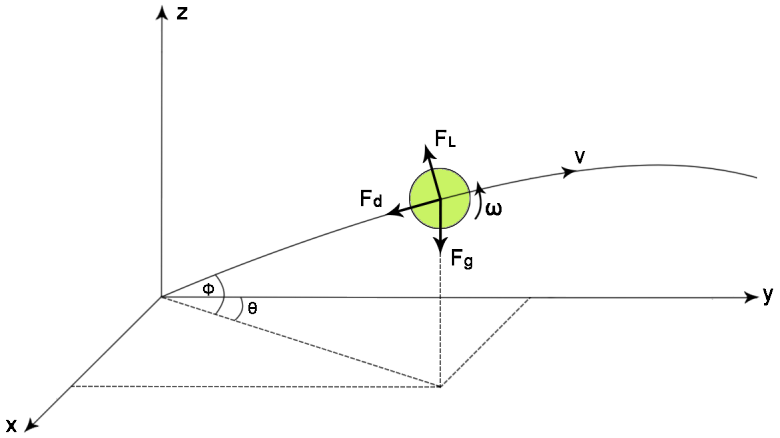

Figure 6: Free Body Diagram of Tennis Ball Forces

Several constraints were used while developing a ball motion model. Firstly, the ball spin axis is assumed to lie parallel to the xy plane (topspin or backspin), due to the limitations of the prototype tennis trainer. An additional wind force was also included in the model, which is assumed to act parallel to the xy plane and is included in the drag force during calculations. For a full derivation of the ball motion model, see Appendix B.

Quadratic drag is used to calculate the drag force due to the high Reynolds number (>1000) associated with tennis balls at maximum velocity [14], see Eq 9. Where v is the relative air velocity, D is the ball diameter and ν is the kinematic viscosity of air.

$$Re = \frac{vD}{\nu} = \frac{30 \cdot 0.066}{1.48 \times 10^{-5}} = 1.38 \times 10^5 \tag{9}$$

Each of the three forces acting on the ball can then be found using Eq 10. Where c_d is the drag coefficient, c_L is the lift coefficient, A is the ball surface area, d is the air density, m is the ball mass, g is acceleration due to gravity and v is the ball velocity relative to the air surrounding it.

$$F_d = \frac{c_d A d v^2}{2} \qquad F_L = \frac{c_L A d v^2}{2} \qquad F_g = mg \tag{10}$$

The lift coefficient can be calculated using Eq 11, where v_{spin} is the rotational velocity of the tennis ball in m/s. c_L is positive for backspin and negative for topspin. Additionally, the drag coefficient is affected by the ball spin and can be found using Eq 12, which is a function that has been fit to experimental data as shown in [14].

$$C_L = \frac{1}{2 + \frac{v}{v_{spin}}} \tag{11}$$

$$C_d = 0.55 + \frac{1}{[22.5 + 4.2(\frac{v}{v_{spin}})^{2.5}]^{0.4}} \tag{12}$$

The final ball motion model, see Appendix B, was simulated at various ball spin velocities using a numerical integration method, see Figure 7. The results from the numerical solution agree with those presented in [14], with an error of approximately 0.5%.

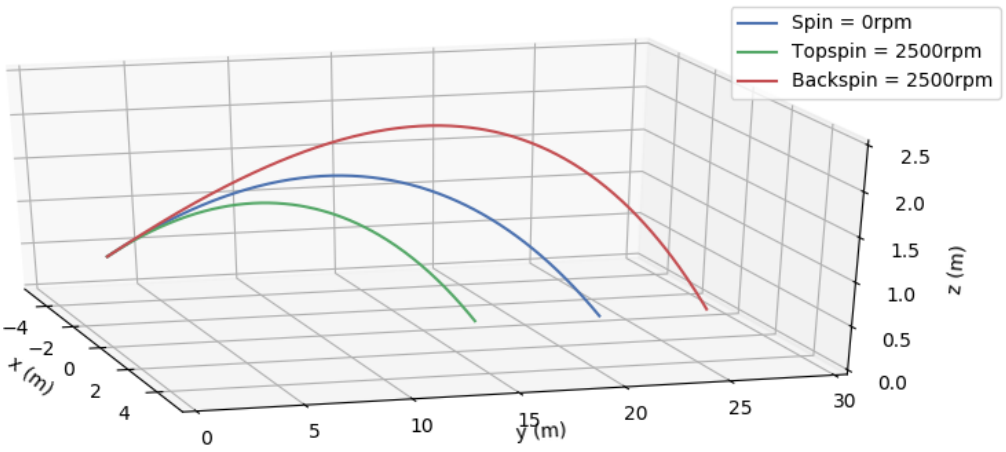

Figure 7: Effect of Backspin/Topspin on Tennis Ball Motion

3 System Overview

3.1 Hardware Selection and Configuration

There are several considerations that need to be acknowledged when selecting suitable hardware for this project. Due to relatively high ball velocities up to 110 km/h (≈ 30 m/s), frame rate has a significant impact on the temporal resolution of ball trajectory data points. Additionally, camera resolution and angle of view (AOV) have an effect on the time required to process each frame and also play a significant role in the successful identification and tracking of ball candidates, as balls can be as small as 3 pixels on the far side of the court. Based on the results of the real-time ball tracking method shown by Polceanu, Petac, Lebsir, *et al.* [6], a maximum processed frame rate of 80fps at 1280x720 resolution could be expected on moderate consumer level processing hardware.

Both single camera and multi-camera tennis ball tracking systems have been the focus of research efforts in the past. The advantage of a multi-camera system the ability to transform points from the 2D image plane to 3D space using epipolar geometry to triangulate points. Advanced ball tracking systems can utilise up to 10 fixed cameras around the court, as discussed in Section 2.1. Using a single camera, the image plane can only be transformed to a single plane using perspective transformation, typically the court area. This allows for basic bounce detection, but it is not possible to accurately estimate the 3D ball coordinates, due to the relatively small ball size and low camera resolution.

Adequate ball bounce detection results were achieved in [6] using a single camera with a fish eye lens at 1280x720 resolution and 25fps mounted at the side of the court. However, due to the low resolution and wide AOV, the ball was occasionally not visible in the image, resulting in a maximum bounce detection rate of 90% on an indoor court.

Frame synchronisation between cameras in a multi-camera system is required for accurate stereo correspondence, however synchronised points can be obtained via linear interpolation where frame accurate synchronization is not possible [7]. A novel approach for frame synchronisation between multiple single board computers (SBC) using a local area network (LAN) and TCP/IP communication is shown in [15].

Single Board Computers (SBC)

Image processing typically requires both significant CPU and RAM resources, due to the relatively large data size of uncompressed images. Additionally, to minimise impact

on the CPU it is desirable to have a direct interface between the GPU and camera for relatively high frame rate image capture. There are two widely available, low cost SBC that fulfil these requirements, the Raspberry Pi 4, see Figure 8, and NVIDIA Jetson Nano. Both of these boards have dedicated GPUs and a single MIPI CSI-2 camera interface, which allows 1Gbps data transfer directly to the SBC GPU, reducing the impact on the CPU.

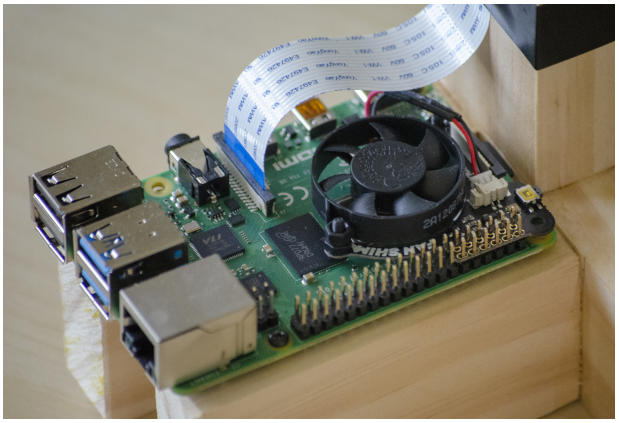

Figure 8: Raspberry Pi 4B

The two boards achieve similar results in benchmark tests [16], with the Jetson Nano performing better in GPU intensive tasks and the Raspberry Pi 4 performing better in CPU intensive tasks. As the Raspberry Pi 4 is significantly more mainstream, there is a much more active community and more supporting resources available.

Camera Requirements

There are numerous USB industrial cameras that can achieve over 300fps and allow synchronisation of cameras in multi-camera systems. However, as this project aims to be relatively low cost, these cameras are far beyond the budget allocation for this project with each camera typically priced between $300 and $1000 AUD.

There are also many board cameras available that can achieve frame rates between 90 and 120fps. Due to the limited processing power available on current generation single board computers, it is desirable to use the MIPI CSI-2 camera interface over USB, as the MIPI interface is connected directly the the GPU with a dedicated processing pipeline, see Appendix C.

Due to the constraints imposed by the Raspberry Pi 4 GPU interface, only Raspberry Pi cameras V1 and V2, see Figure 9, can be used with the SBC. These are high quality

single board cameras, capable of recording 1280x720 video at up to 90fps. One limitation of the Raspberry Pi cameras is the inability to synchronise cameras directly. However, this can be overcome using synchronisation over TCP/IP, as discussed in Section 3.1.

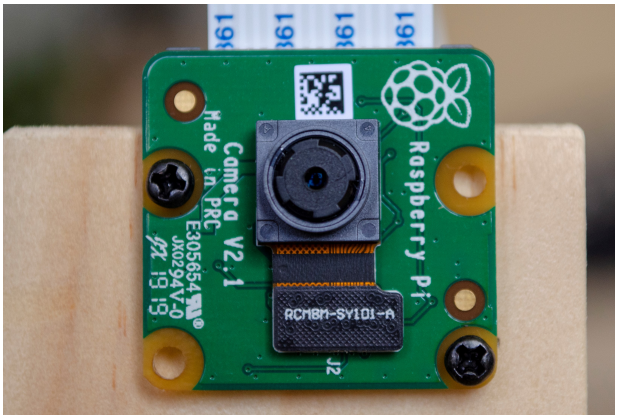

Figure 9: Raspberry Pi Cam V2

Stereo Cameras

The depth resolution of the stereo cameras can be found using Eq 13, where δz minimum change in depth, z is the depth, B is the baseline and f is the focal length in pixels. As δz increases, there is lower depth resolution, therefore the resolution decreases with the square of the depth. To increase the depth resolution, the baseline can be increased, however this requires the stereo cameras to be tilted inward, decreasing the area covered by both cameras. It is therefore desirable to balance these two competing factors to achieve acceptable resolution while maintaining camera coverage.

$$\delta z = \frac{z^2}{Bf} \tag{13}$$

The largest baseline which maintained adequate stereo coverage was found to be 4m, see Appendix D. This relatively large baseline enables sufficient disparity between images, while remaining small enough to be rigidly mounted to the tennis trainer machine.

The focal length can be obtained from the camera calibration intrinsic matrix, which was found to be 1.24×10^3 pixels in the case of the Picamera v2. Solving Eq 13 with a 4m baseline and a 24m depth, approximately the length of a tennis court, the maximum achievable resolution per pixel is estimated at 11.6cm. Although this is quite high

compared to the 6.6cm diameter of a tennis ball, it was deemed acceptable as an upper bound at the full court length for the prototype system.

Using a larger baseline to increase depth resolution is certainly possible with this configuration. However, this would require either rigid mounting of cameras to the tennis court, restricting the portability of the tennis trainer machine, or would require dynamic calibration techniques for portable untethered cameras. The challenges involved with dynamic stereo calibration are outlined in Section 4.

A small stereo camera prototype was constructed to allow for testing and evaluation of stereo computer vision techniques, see Figure 10. Each camera is connected to a dedicated Raspberry Pi for image capture and image processing, with the two cameras placed 400mm apart (1:10 scale) to provide adequate feature disparity for preliminary testing and evaluation.

Figure 10: Stereo Camera Prototype

The standard Picamera v2 has an effective angle of view (AOV) of 51° when using the desired 1280x720 resolution [17], giving the 6.6cm ball an effective pixel resolution of 3.6 pixels on the far side of the court. However, as discussed in Section 5, this resolution was later reduced to 640x480 due to limitations in the practical capture speed of the Picamera v2. This also reduced the effective AOV to 28°, while maintaining the same effective ball size of 3.6 pixels on the far side of the court.

Two mounts were designed and 3D printed which would hold the Raspberry Pi 4 boards

and the cameras, see Figure 11. The face which holds the camera tilts downwards at 10° and inwards at 6°, to ensure the AOV from both cameras overlap the desired court area. The mounts were designed to fit on a 19x42mm timber frame and allow for independent horizontal adjustment of each camera.

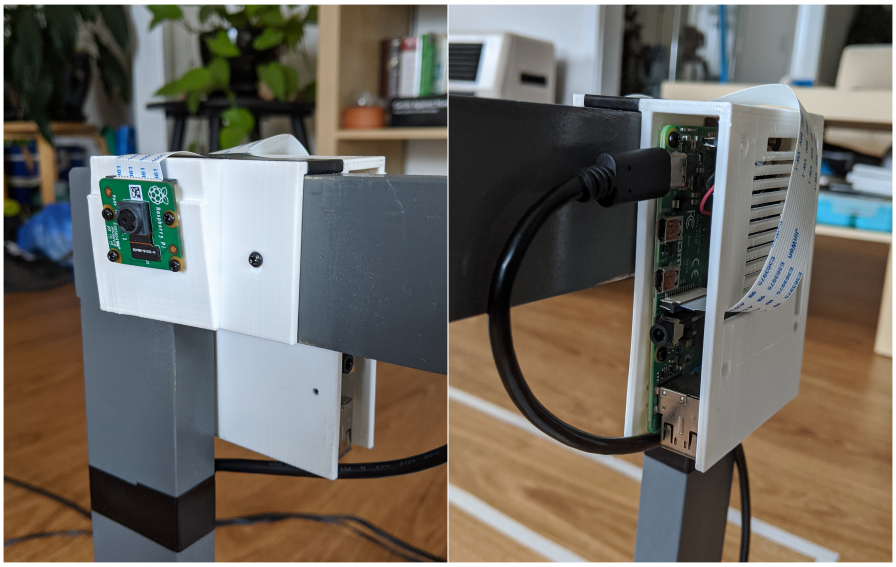

Figure 11: Right Camera and Raspberry Pi Mount

3.2 Programming Language and Utilisation of Libraries

The two programming languages considered for this project were C++ and Python 3.7. Both are object oriented languages with extensive resources and support available online. They are also both supported on the Raspberry Pi Linux distribution Raspbian Stretch, which is being used for this project.

C++ offers superior computational performance over Python, largely due to the the use of fixed data types and pre-compilation of code prior to runtime, in contrast to Python using dynamic typing and code being interpreted at runtime. However, Python offers significant advantages during development over C++ for several reasons.

Firstly, it has a vast range of open source libraries available, often with quick and simple installation available via the Python package manager PIP. Additionally, many of these libraries are written and compiled in C or C++ and wrapped in Python, giving the performance advantages of C/C++ with the ease of use of Python. Development using Python is also usually accelerated due to its simple syntax, making it ideal for rapid prototyping applications.

Additionally, as Python does not need to be compiled prior to runtime, it is ideal when development is occurring over multiple target machines and operating systems, as is the case with this project.

Numpy and OpenCV

The Numpy and OpenCV Python 3.7 libraries are used extensively in this project. Numpy is a collection of high level mathematical functions, with support for multi-dimensional arrays and matrices, where OpenCV is an open source computer vision toolbox. Both are compiled in C/C++, giving significant performance advantages over comparable implementations directly in Python. As OpenCV is designed to give very generalised implementations of many computer vision algorithms, it should be noted that this often comes with a significant performance cost. After prototyping and design is complete, performance gains could likely be achieved by simplifying these algorithms for the given application.

Multi-processing

One caveat of Python is the global interpreter lock (GIL), which prevents multiple threads accessing the python interpreter simultaneously and restricting execution to a single CPU core. Multi-processing is a python library which implements a workaround for the GIL, essentially enabling multi-threading behaviour. Multi-processing differs from multi-threading in that it spawns a completely separate python interpreter, with a completely separate memory space available. This allows concurrent processing, however the use of shared memory spaces common in multi-threading applications must be defined explicitly. Fortunately, multi-processing has several of these explicit shared objects protected by mutex locking and unlocking, Value, which can be any standard c type, Array, which is an array of c types and queues, which are essentially message oriented sockets.

3.3 Communication and Networking

Each camera is connected to a dedicated Raspberry Pi SBC, with the main tennis trainer program and stereo reconstruction being executed on a third computer. To enable a mechanism for high speed, reliable communication between the three devices, Python functions have been developed using network sockets for communication using TCP/IP over a local area network (LAN) or wireless local area network (WLAN).

The main computer acts as a server, with each camera computer connected as a client over the network. Several functions have been developed that allow for any Python

object to be compressed and sent over the network. Firstly, the data to be transferred is converted to a serial byte array using the *pickle* module, which is a built in module in Python 3.7. A fixed length header is then constructed, which contains the length of the serialised data, stored as a string encoded using UTF-8 to ensure cross platform compatibility. The data is appended to the end of the header, which is then sent to the server via the relevant socket, established using the server local IP address.

Once the corresponding client socket on the server indicates that a message has been received, a number of bytes equal to the fixed length header are read from the socket. This string is decoded and converted to an integer, defining the length of the message. The number of bytes equal to the message length is then read from the socket, which can then be de-serialised to recover the initial data.

A custom data structure was developed and used extensively throughout this project for communication between the server and clients. The data structure contains both a string representing the type of data being sent followed by the relevant message data. The type of message can be commands, such as 'record' or 'shutdown', ball data containing the image coordinates of balls in a frame or image data, containing the raw data from the entire frame.

3.4 Generation of Test Data

Ideally, while developing the computer vision algorithms for ball detection and tracking, real video of tennis shots would be used to evaluate performance in real conditions. However, to increase the flexibility of the gathered data and speed up the prototyping phase of the software development, a 3D model of a tennis court was created using Blender. Cameras can be placed anywhere in the scene, with intrinsic camera parameter settings available for matching the properties of real cameras. Physics simulations can also be conducted in Blender and used to animate the scene using a realistic tennis ball model, see Figure 12. Care needs to be taken when using this data for testing however, as real images and scenes are much more complex.

The coefficient of drag used in Blender simulations was estimated experimentally using the results from [14], which indicates that a ball hit from 1m above the baseline at 30m/sec with no spin at an angle of 8.1° should land on the opposing baseline. Ball bounciness was also experimentally calibrated using the ITF standards [18], which detail that a ball dropped from 2.54m must bounce 1.35 to 1.47m high.

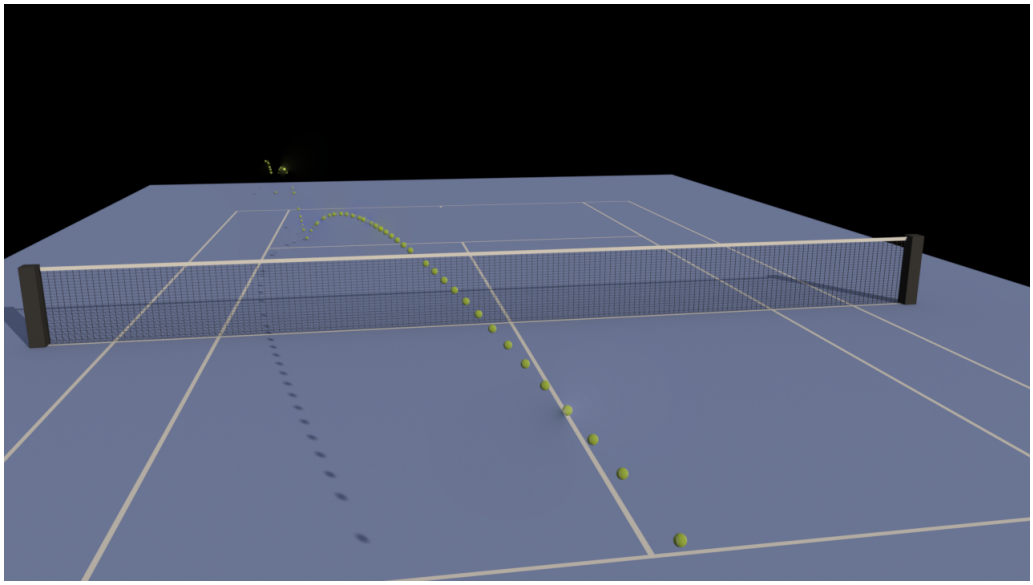

Figure 12: Tennis Court 3D Render

3.5 Computer Vision Pipeline

There are a number of steps that must occur sequentially for each frame in order to obtain the 3D coordinates of the tennis ball in world coordinates.

1. **Image Capture**
   Grayscale images are captured from both cameras simultaneously.

2. **Background Subtraction**
   Captured images are compared with mean and standard deviation images previously computed. New foreground objects are the only objects remaining after subtraction.

3. **Object Detection**
   Each image is searched to find remaining foreground pixels that are connected. Each object is labelled and object size, width, height and x,y coordinates are recorded.

4. **Rectification**
   Coordinates of each object are rectified, meaning they are transformed using the relationship between the two stereo cameras so that both image planes are coplanar.

5. **Feature Correspondence**
   Objects from the left image are compared to objects in the right image, objects are then matched using various object features.

6. **Triangulation**
   Matched rectified object coordinates are triangulated, returning the object position in 3D world coordinates.

3.6 Program and Processing Pipeline

There are two primary programs that control the operation of this system. The client program, see Figure 13, runs on both Raspberry Pis. It sends and receives messages to and from the server, controls the Picamera operation and conducts all computer vision steps before rectification, see Section 3.5. To enable the utilization of multiple processor cores of the raspberry Pi, the program is split into n+2 separate processes, indicated by the vertical grey dotted lines, where n is the number of processes used for image processing.

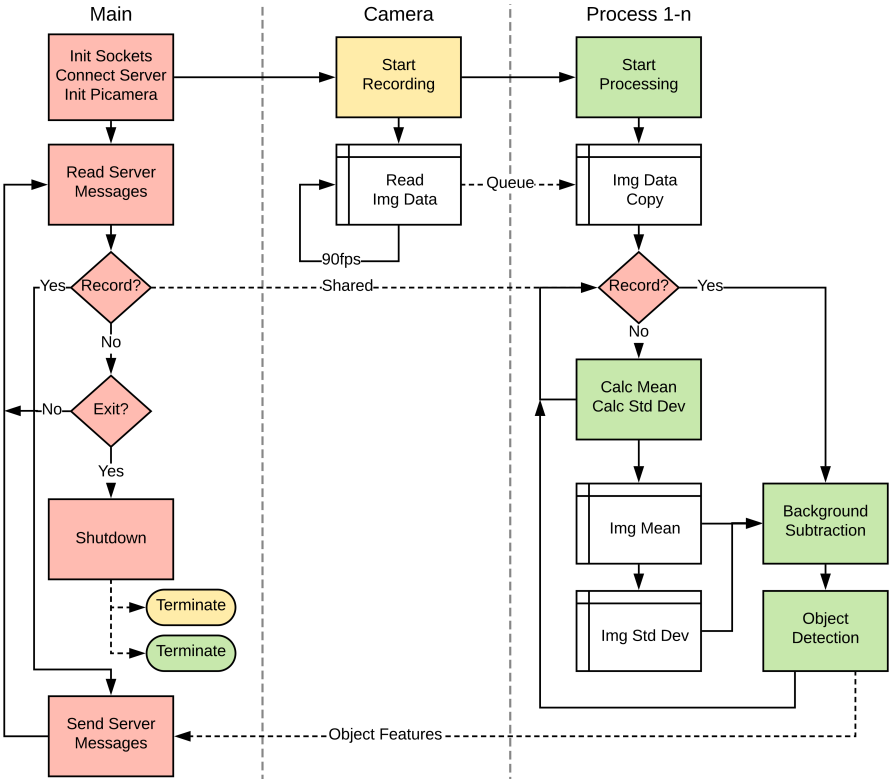

Figure 13: Client Program Flowchart

The main process manages all incoming and outgoing messages to the server and also manages all other processes. The camera process manages the Picamera operation, ensuring that image data is read from the associated image buffer fast enough to maintain a high frame rate without dropping frames, as there is no interrupt generated by the camera when new image data is available. This requires the camera process to cycle in less time than it takes to capture the next frame to avoid dropping frames [17].

All additional processes read image data from an image queue shared with the camera process, images are then processed accordingly depending on the current state of the system. Objects found within each frame are then sent back to the main process for transmission to the server.

The server program runs on the tennis trainer and controls when to start/stop recording, see Figure 14. This program manages client connections and completes computer vision processing steps after object detection, see Section 3.5. While recording, all computer vision steps are happening sequentially for each frame, but in parallel for the whole recording, i.e. the server does not need to wait for all frames to be processed before rectification, correspondence and triangulation can occur. This is essential for near real-time performance.

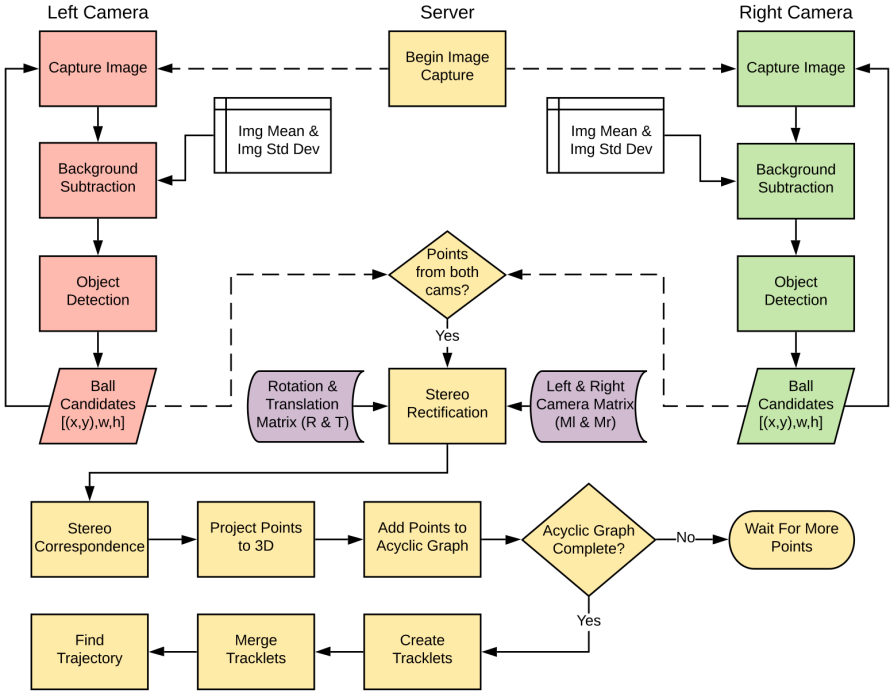

Figure 14: Server Program Flowchart

4 Stereo Calibration

Stereo calibration is the process of determining the spatial relationship between a stereo camera pair, as presented in Section 2.4. Typically, stereo calibration is conducted using multiple views of a planar calibration object, such as the chessboard pattern seen in Figure 3. However, for stereo cameras with a large baseline and narrow angle of view, as in the case of this project, using a calibration object such as a chessboard is not practical, due to the large and cumbersome size of the object that would be required.

The clear candidate for stereo calibration in this case is therefore to use the tennis court lines for calibration as tennis court lines are standardised in terms of colour (white), size (5-10cm) and layout, making detection of lines relatively straight forward. However, using a calibration object provides many views of the same planar object, allowing the rotation and translation matrices (R & T) to be estimated based on a set of possible solutions. Whereas the tennis court lines on a planar surface only provide one view and therefore one solution for R & T, resulting in a relatively poor stereo calibration with a high reprojection error and sensitivity to outliers.

4.1 Stereo Calibration Methods

There are several methods which could be used for stereo calibration in the context of this project, depending on the portability required of the implementation.

Fixed Implementation
If the stereo cameras are permanently fixed to the court, calibration can still be achieved using a large stereo calibration pattern as the calibration only needs to be conducted once and therefore using a large calibration pattern is not entirely impractical. Additionally, the camera rotation and translation matrices could be estimated using traditional measuring techniques and hard coded into the system. However, the clear limitation to this method is that the tennis trainer can only be used in one location, which is likely too prohibitive for most use cases.

Cameras could also be fixed relative to one another, but still remain portable with the tennis trainer machine. However, as the 4m distance between cameras is relatively large, this still significantly limits the portability of the camera system and tennis trainer.

Dynamic Implementation - Calibration Using Rich Feature Descriptors
Rich feature descriptors are essentially points within an image which are described by their surrounding environment. Feature extraction and matching is a common operation

in computer vision and robust algorithms exist for this purpose, such as the Speeded-Up Robust Features (SURF) algorithm. After feature matching, the fundamental matrix (F) can be estimated using the 8-point algorithm. From this, the essential matrix (E) can be found using the relationship to the camera matrix shown in Eq 14, which then contains all the information about the rotation and translation between cameras in world coordinates. This method would allow the cameras to be independent of one another and therefore significantly increases the portability of the system.

$$E = M_l^T F M_l \tag{14}$$

4.2 Prototype Stereo Calibration Implementation

As discussed, a robust stereo camera calibration is crucial for triangulation of ball coordinates resulting in the entire vision system prototype resting on this calibration. For this reason, it was determined that to speed up the development of all subsequent system components, stereo calibration using a planar calibration object would be acceptable for this prototype.

Additionally, using a fixed calibration determined using a calibration object allows for testing to be conducted in a controlled test environment. If a tennis court is required for stereo calibration whenever a test must be performed during development, this significantly limits the available testing locations, increases development time and is not easily demonstrable in a laboratory environment, as is required upon completion of this prototype.

5 Image Capture

5.1 Resolution and Frame rate

The desired frame rate of this system is 90fps, to ensure maximum temporal resolution of 3D points to determine ball trajectory. As discussed in Section 3.6, the process responsible for reading image data from the camera buffer, and subsequently writing that data to a shared multiprocessing queue, must complete each cycle in less time than it takes for each frame to be captured (11.1ms for 90fps). To ensure consistent frame rates, the camera is configured to record 90fps video, with the output from the image signal processor (ISP, see Appendix C) copied to a buffer in 24-bit YUV format (brightness, red projection, blue projection) before any JPEG encoding occurs. The Y component can subsequently be read directly from this buffer to obtain the grayscale image.

Initially, the camera was configured to record at 90fps using 1280x720 resolution, as per the Picamera documentation [17]. However, after implementation it was observed that only approximately 30-40 fps could be achieved at this resolution without many dropped frames.

Upon further investigation, it is clear why recording uncompressed video at 90fps is not possible using this configuration. Typical video recording and subsequent encoding is performed entirely within hardware on the GPU, See Appendix C, after which the compressed video data is copied to the CPUs RAM via direct memory access (DMA). However, when reading each individual frame from the camera while video recording, the uncompressed YUV data is transferred directly to the CPUs RAM via DMA before encoding.

The amount of data per frame can be calculated by multiplying the number of pixels in the image by 24 bits per pixel. The data per frame is therefore 22.1Mb for 1280x720 resolution and 7.4Mb for 640x480 resolution. At 90fps, this amounts to a required data transfer speed of 2Gbps for 1280x720 resolution and 664Mbps for 640x480. Assuming the DMA is operating in burst mode, transferring 1 byte of data requires 8 clock cycles. As the Raspberry Pi 4 has a default clock speed of 1.5GHz, this provides an estimate for the maximum possible DMA transfer speed of 1.5Gbps, which is not sufficient to capture the full 1280x720 resolution at 90fps in uncompressed 24-bit YUV format.

This has significant implications for the use of Picameras in this project. Firstly, experiments confirmed that 90fps video recording was achievable at a lower resolution

(640x480). However, this change in resolution does not simply scale the image previously captured in 1280x720. It instead uses a smaller area of the image sensor, effectively reducing the angle of view AOV to approximately 28°, see Figure 15.

Figure 15: Picamera V2 Resolution and AOV [17]

Fortunately, the decreased AOV has the effect of maintaining the original size of image features, such that the ball still occupies just over 3 pixels at the full court range, the same as when using 1280x720 resolution. However, the decreased AOV also has the effect of significantly reducing the area covered by both cameras and also reduces the number of court features visible in the image. The implications of this on stereo calibration are discussed in Section 4.

5.2 Frame Synchronisation

For stereo vision systems to work well, images need to be captured at precisely the same time, else complex and computationally intensive inter-frame interpolation techniques must be conducted. As there is no hardware frame synchronisation feature available on the Picamera, synchronisation must be achieved in software. Once the Picamera

is initialised in video recording mode, it is always recording and there is no way to synchronise the start of each frame precisely. However, given a consistent 90fps frame rate, as long as the captures begin at precisely the same time, they will only differ by a maximum of 1/2 of the time taken to capture 1 frame (5.5ms).

The implementation of network sockets to communicate between devices in this system was designed to achieve this synchronisation. Effectively, once cameras have been initialised and relevant processes started in the client programs, each client simply waits for a 'record' message from the server. Over a local area network with relatively low latency, this communication and subsequent processing typically occurs in several milliseconds.

Tests were conducted to ensure frame synchronisation was achieved between cameras. A digital stopwatch with millisecond display was set up on a monitor with a 144Hz refresh rate. Ideally, an analog method of calibration would be used, however as the 144Hz refresh rate of the monitor is faster than the 90fps recorded by the camera, any difference between frames should still be visible. The tests showed both cameras were synchronised to within 6.9ms, given the 144Hz refresh rate, see Figure 16.

Figure 16: Frame Synchronisation

This level of synchronisation was deemed to be acceptable for prototype testing and evaluation, given the controlled state of the LAN. However, using this method for synchronisation is susceptible to significant error if the network latency is not low and consistent between clients. Other methods of synchronisation via network time protocol (NTP) provide possible solutions this latency sensitivity, though would require an active internet connection on the network for synchronisation with NTP servers. Alternatively, the two clients could be synchronised electrically via a connection between the clients and server, though this is also undesirable due to the additional hardware required.

6 Background Subtraction and Object Detection

6.1 Background Subtraction

A background subtraction method using single Gaussian image differencing, as presented in [8], was implemented in Python. Firstly, grayscale images are used to reduce computational complexity, with each pixel represented as an 8-bit intensity value between 0 and 255. Background pixel intensities are modelled as a single Gaussian, each with a mean (μ) and standard deviation (σ) that is updated each frame. Pixels that are found to be outside of three standard deviations from the mean are considered foreground pixels. A learning rate (ρ) is also included to control the rate at which the background Gaussians update each frame, which is adjusted depending on how dynamic the background of the scene is. Gaussians are updated using Eq 15 & 16, where I_t is the pixel intensity at time t.

$$\mu_t = (1-\rho)\mu_{t-1} + \rho I_t \tag{15}$$

$$\sigma^2 = (1-\rho)\sigma^2_{t-1} + \rho(I_t - \mu_t)^2 \tag{16}$$

Differencing between the foreground of the previous frame and the foreground of the current frame results in a set of candidates S_1, the current foreground gives another set S_2 and a third set S_3 is obtained by the logical and of S_1 & S_2. S_3 should then only contain pixels that are foreground pixels and are new in the current frame, assuming that the ball has moved to an entirely different set of pixels between frames, due to it's relatively high velocity. Preliminary background subtraction results are shown in Figure 17.

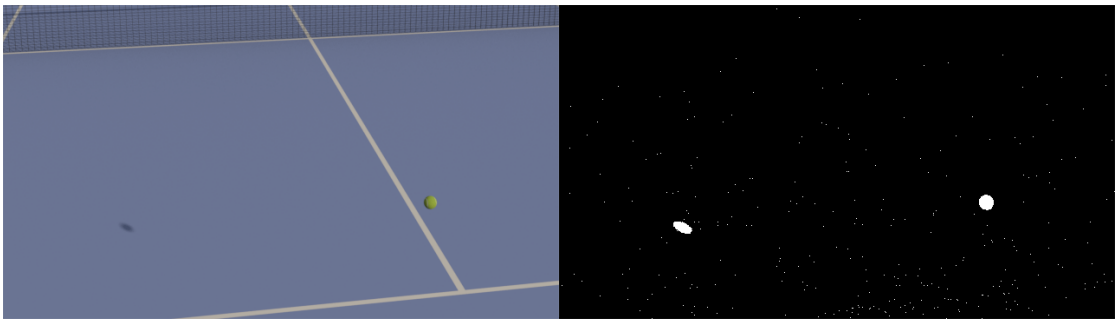

Figure 17: Background Subtraction Output (simulation)

6.2 Object Detection

Initially, an algorithm for labelling foreground objects was developed that scans through the image using a pre defined structure, see St_1 in Eq 17, where the current pixel is the element (2,2). Ones in St_1 represent the neighbouring pixels to be checked for existing labels, if a label exists in either of those pixels, it is connected to the current pixel. Connected objects were then linked together using inheritance and the final object properties stored in an array. This method performed adequately on test data, though inheritance occasionally failed on complex shapes.

The processing speed of this algorithm was tested against labelling functions in existing image processing libraries, such as *connectedComponentsWithStats()* in OpenCV [19]. The OpenCV label algorithm performed approximately 15 times faster than the custom algorithm, due to it's implementation being highly optimised using C code. The OpenCV label algorithm is similar to the one developed previously, however it utilises a different structuring element, see St_2 in Eq 17, to scan the image during initial labelling. Once two different labelled objects connect, the algorithm looks up the root location of the connecting label in an array and rescans the object, merging the two labelled regions.

$$St_1 = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \qquad St_2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{17}$$

After labelling, the same function in OpenCV is used to obtain object parameters such as coordinates, width, height and size. If the object size and shape are within bounds, the object is deemed to be a valid ball candidate and its details stored in an array of ball candidates. As seen in Figure 18, this method is able to successfully detect the tennis ball, though also detects the ball's shadow as a candidate. This is acceptable, as non-ball objects will be removed by the trajectory detection algorithm.

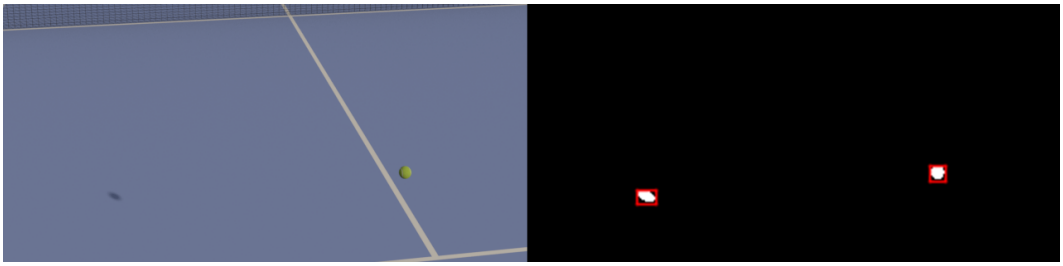

Figure 18: Filtered Ball Candidates (simulation)

6.3 Image Noise

There are several types of noise present in digital images, these include random noise, banding noise and fixed pattern noise [20]. This noise is primarily caused by electronic noise on the analog to digital converter (ADC) circuit in the image sensor, though is also influenced by the ISO, a gain which controls the sensitivity to brightness of the camera. A low ISO will produce darker images with less noise, a high ISO will produce brighter images with more noise. Due to the high frame rate required for this system, a short exposure time and therefore relatively high ISO will need to be used.

Random noise has significant implications for this project, primarily during the background subtraction stage of the processing pipeline. As the luma or brightness component of the video is the one being utilised for image capture, the image noise significantly influences the grayscale pixel values. Fortunately, the Gaussian image differencing method inherently removes random noise from the mean and standard deviation reference images, as the method requires changes in the image to persist over a period of time controlled by the learning rate, see Section 6.1.

However, this random noise is especially prominent in the images that are being compared to the mean and standard deviation images. This results in many more single pixels being detected as foreground objects and has a detrimental effect to later object detection and processing stages. Three different methods were examined to attempt to reduce the impact of this noise.

1. Removal of small objects after object detection
This method ignored the noise present in the image, allowing all pixels to be detected as foreground objects and subsequently processed. After detecting objects, objects with a size smaller than a threshold size would be removed from the candidate object list.

This method proved to be very effective during initial testing, removing the majority of the noise from the image and leaving the ball and other large objects unchanged. However, it proved to be extremely slow, as the number of detected objects often exceeded 10000, being highly dependent on the amount of noise present.

2. Erosion and Dilation of binary images
Erosion is a morphological image processing technique that operates on binary images. A structuring element or kernel is passed over the binary image, with the origin of the kernel passing over each pixel. If the kernel is completely contained in the region of the target image, the pixel at the origin is retained, else it is discarded. For example in

Eq 18, if the kernel A with origin at [0,0] is applied to the target matrix B, C is the resulting eroded matrix.

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \qquad C = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{18}$$

Dilation is another morphological image processing technique which is effectively the inverse of erosion. It operates using the same structuring element technique described previously, except it fills in missing pixels in the target image instead of removing them when the origin of the kernel is on an existing pixel. However, although dilation is the inverse of erosion, it does not perfectly restore details which were previously lost in the erosion step. For example, following the previous example, see Eq 19. After the dilation operation is applied to B, the resulting matrix C is significantly different from the original matrix B in Eq 18.

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad C = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \tag{19}$$

This loss of detailed object information is important to keep in mind when using these operations, particularly in a stereo imaging application, where the difference of several pixels can have a significant effect on the accuracy of the final object triangulation calculations.

3. Image Filtering Using a Low Pass Filter (LPF)

A low pass filter in the context of image processing aims to smooth out rapid changes in pixel intensity. This is achieved by calculating the average value of the surrounding pixels located within the area of a specified kernel and replacing the target pixel with that average value. For example, in Eq 20 a kernel A with origin [0,0] is passed over the matrix B, with the resulting smoothed matrix C. This has the effect of scaling the intensity of small groups of pixels according to their size. A simple threshold can then

be performed on the image, retaining pixels over a certain value.

$$A = \begin{bmatrix} \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad C = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.25 & 0.5 & 0.5 & 0.25 & 0 \\ 0 & 0.5 & 1 & 1 & 0.5 & 0 \\ 0 & 0.5 & 1 & 1 & 0.5 & 0 \\ 0 & 0.25 & 0.5 & 0.5 & 0.25 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{20}$$

The computational cost of this method scales linearly with the size of the kernel being used, given that more calculations must be performed for a larger kernel.

6.4 Testing and Evaluation of Noise Reduction Methods

Firstly, a 3 second video was recorded at a resolution of 640x480 and a frame rate of 90fps. All video frames were saved, alongside the mean and std deviation images calculated prior to recording. A script was then developed to gather the ground truth ball location data manually. Each video frame was displayed on screen until a mouse click was recorded, after which the coordinates of the mouse pointer were saved as the ball coordinates for that frame and the next frame displayed.

The location of this test was chosen to examine various aspects of the video without repeating the time consuming task of manual ground truth labelling, see Figure 19.

Figure 19: Test Location

The scene contains significant dynamic background changes due to the motion of distant trees, contains both shaded and brightly illuminated areas and also covers a distance of approximately 25m to examine the detection of the ball at greater distances. The ball was thrown at approximately 50 km/h away from the camera.

A test script was then developed to imitate the operation of the camera. Frames were read in by one process and added to a thread safe queue. Four processes were then created to process the frames from the queue using the noise reduction and background subtraction method being assessed, with resulting ball candidates appended to an output queue. Replicating the behaviour of running multiple processes simultaneously was crucial to evaluate different methods, as frames are not neccessarily processed sequentially or in the correct order by the main camera script.

The test script was also designed to generate a small .mp4 video of the original frames and the resulting background subtracted frames, with ball candidates overlayed as red x marks. This enabled quick visual comparison between tests during evaluation.

After ball candidate detection, the test script compared all ball candidates in each frame with the ground truth ball coordinates recorded previously. Ball candidates within a 5 pixel radius of the ground truth were recorded as true positives, where all candidates outside this radius recorded as false positives. The processing time for each frame by each process was also recorded, though it should be noted that this analysis was not performed on the target hardware and is therefore only useful for relative comparison between tests.

Tests Conducted

Firstly, identification of independent variables in each noise filtering technique was conducted. These variables include the kernel shapes and sizes used for erosion, dilation and filtering, the order these operations are conducted and the threshold values for low pass filtering and ball size filtering. Additionally, the effect of differencing the current foreground from the previous foreground, as described in Section 6.1, was examined.

Visual analysis of the resulting videos was first conducted to observe any potential candidates for further analysis. Seven tests were then conducted with the following properties:

1. Ball size $>$ 5px, No foreground difference (control)
2. Ball size $>$ 5px, Foreground difference
3. Ball size $>$ 4px, Foreground difference, 5x5 kernel, LPF threshold 174

4. Ball size > 4px, Foreground difference, 4x4 kernel, LPF threshold 174
5. Ball size > 2px, Foreground difference, 4x4 kernel, LPF threshold 150
6. Ball size > 1px, Foreground difference, 3x3 + kernel close, 5x5 circle kernel erode
7. Ball size > 6px, Foreground difference, 2x2 kernel erode, 2x2 kernel dilate

Results and Discussion

Several criteria were examined to determine the efficacy of each noise reduction technique.

- The true positive rate (TPr) is defined as the ratio of valid ball candidates to ground truth balls in the recording.
- The average false positive count (FPμ) is the average number of invalid ball candidates per frame.
- The processing time (t) is the time taken for a frame to process in ms.

Therefore, true positive rate should aim to be maximised, false positives minimised and processing time minimised, see Table 1 for results.

Table 1: Noise Filtering Results

Test	**t(ms)**	**TPr**	**FPμ**
Test 1	12.90	100%	447.38
Test 2	10.00	99%	131.14
Test 3	7.25	72%	0.07
Test 4	6.79	81%	0.16
Test 5	6.83	86%	1.36
Test 6	5.67	74%	0.85
Test 7	5.56	90%	7.91

From Table 1, it is clear that the configurations in Test 1 and Test 2 are able to detect the ball very reliably, with a true positive rate of 100% and 99% respectively. However, both tests have an extremely slow processing time relative to other tests and have also have much higher average false positives, see Figure 20 where the ball is indicated by the blue circle and ball candidates by red crosses. However, the effectiveness of foreground differencing is highlighted by the average number of false positives in test 2 being only 29% of that in test 1, a substantial reduction using a simple processing step.

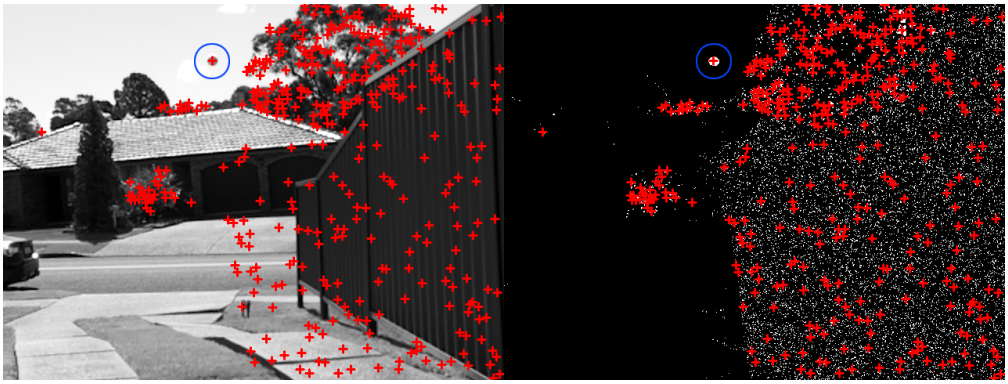

Figure 20: Test 1 - No noise filtering techniques applied (control)

Tests 3,4 and 5 demonstrate that a 4x4 kernel is most effective at removing noise while maintaining a relatively high true positive rate of 86%. This does however come at a cost of 1.36 average false positives detected per frame.

Test 6 and 7 show that erosion with a 2x2 kernel followed by a dilation with a 2x2 kernel is more effective at detecting true positives, with 90% of balls detected. However, there is a substantial increase in average false positives.

From these tests, a reasonable conclusion is that the processing method presented in test 5 is the most effective, see Figure 21. It achieves a true positive rate of 86% in the challenging test conditions presented, only 4% below test 7. The much lower average false positive rate in test 5 of 1.36 is a significant advantage over test 7, as the additional computation required during stereo correspondence, due to the higher number of ball candidates in test 7, will likely make up for the small difference in processing time of 1.27ms.

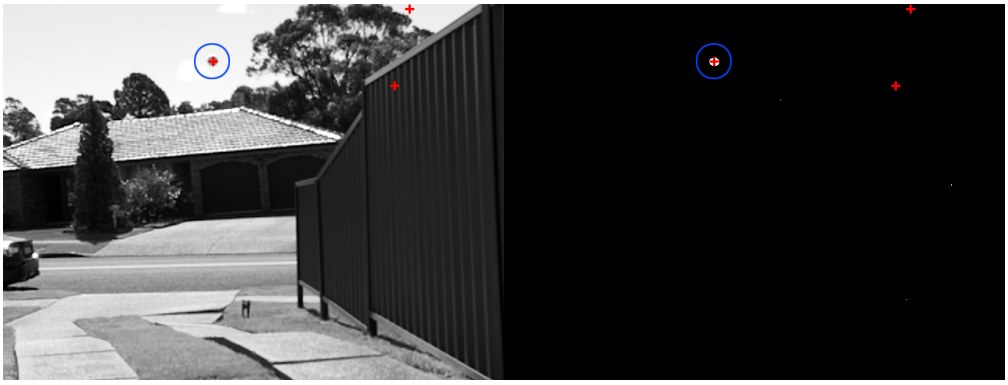

Figure 21: Test 5 - Low pass filtering with 4x4 kernel, threshold value 150

6.5 Sensitivity to Lighting Conditions

One of the major hurdles when implementing any background subtraction algorithm is the sensitivity of the algorithm to lighting conditions present in the scene. In more conventional colour based segmentation methods, the colour of an object can vary drastically as it passes through areas of variable lighting. The Gaussian image differencing technique implemented in this project is somewhat robust to changes in illumination of the target object, as it is simply any change in the pixel brightness that is being detected as a foreground object, not any brightness range in particular.

However, there are still limitations to the technique. For example, an experiment was conducted where an open window allowed bright natural light to flood the scene, see Figure 22. The image overlay in the top right corner is the ROI after background subtraction, the red crosses are objects detected. The blue circle in the image overlay represents the actual position of the ball in the frame.

It is evident in this experiment that detecting the ball in this environment is not feasible with any form of brightness based background subtraction algorithm, as it is barely visible to the human eye.

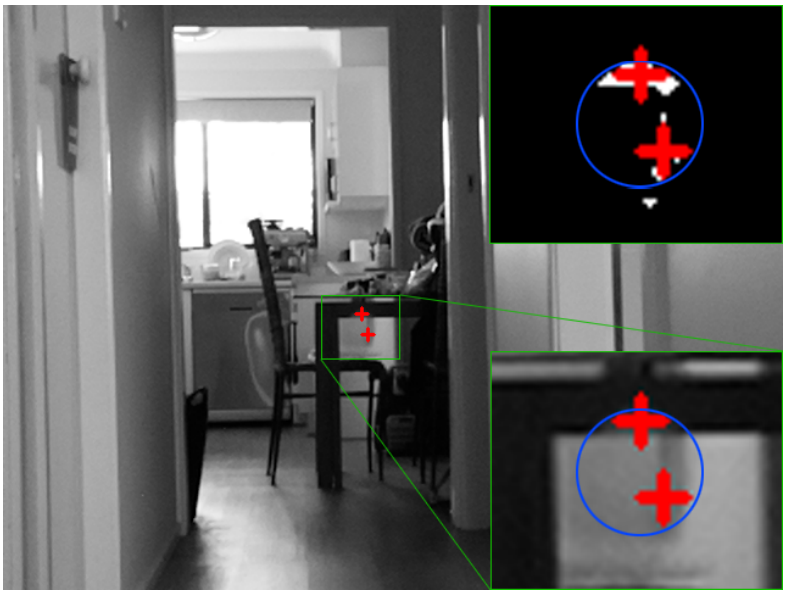

Figure 22: Sensitivity to lighting conditions

There are two ways to reduce the impact of variable lighting conditions during ball detection. Firstly, restricting the use of the system to conditions where there is relatively uniform lighting, i.e. outside on a bright day or inside with no external light sources.

Secondly, the ISO of the camera can be manually adjusted based on the conditions. Allowing the ISO to adjust automatically is usually desirable to get an image that is somewhat balanced in terms of illumination. However, if there is a dominant source of illumination in the scene, the brightness resolution will be subsequently decreased in other areas of the image.

To reduce the added complexity of operation involved with manually adjusting ISO values depending on the operational conditions, it was deemed to be acceptable to operate and test the system only in favourable lighting conditions.

6.6 Foreground Differencing Evaluation

As stated in Section 6.1 and Section 6.4, a key part the implemented background subtraction algorithm is the foreground differencing between frames. To reiterate, during background subtraction, the foreground that is extracted in the current frame is compared to the foreground in the previous frame. Only pixels that are different between frames are considered foreground pixels.

As described in [8], this method rests on the assumption that the tennis ball is moving sufficiently quickly in the frame to occupy an entirely new set of pixels each frame. If for instance, the tennis ball is not moving fast enough within the frame, significant occlusion will occur, see Figure 23, where four sequential frames are shown from left to right. The ball is indicated by the blue circle and the detected object centre by a red cross. It is clear that any significant ball occlusion will tend to offset the centre of the detected object, significantly affecting triangulation results further down the processing pipeline.

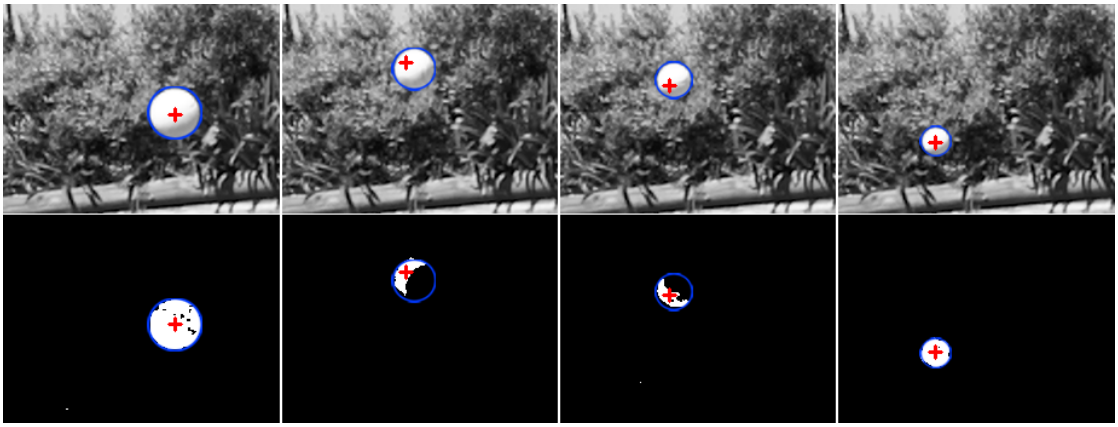

Figure 23: Ball occlusion due to slow relative motion within frame

However, there are several justifications for the continued use of this method for this system. Firstly, as indicated in Section 6.4, foreground differencing is a powerful technique for rejecting false positives, with test 2 showing a 71% reduction in false positives over test 1, which did not use foreground differencing. Additionally, as this system aims to detect balls launched at relatively high velocity from a tennis trainer, the assumption that balls will be travelling at sufficient speed through the frame likely holds.

If in practice this assumption does not hold, the inter-frame foreground differencing spacing can be increased. Practically, with four processes running in parallel, this inter-frame spacing is approximately four frames. It can therefore easily be increased to any multiple of four frames to reduce the likelihood of ball occlusion. Furthermore, the captured video frame rate could be reduced to increase the time between subsequent frames. Reducing the number of recorded frames would also likely lead to a significant increase in overall system speed, albeit at the expense of temporal resolution.

7 Rectification, Stereo Correspondence and Triangulation

7.1 Rectification

As outlined in Section 2.2, rectification is the process of using the camera matrices ($M1$ & $M2$) from camera calibration and rotation and translation matrices (R & T) from stereo calibration to obtain projection matrices ($P1$ & $P2$) and rotation matrices ($R1$ & $R2$) used for stereo correspondence.

To ensure the correct matrices had been obtained during calibration, coordinate maps were generated using the relevant projection and rotation matrices using *initUndistortRectifyMap()* and *remap()* functions available in OpenCV. Horizontal lines were then drawn on the image to verify the rectification visually. All points in the left image should lie on the same horizontal line in the right image, see Figure 24. Note that although the right image appears smaller than the left image, this is simply due to some of the image being mapped into negative pixel coordinates and not displayed. In practice, negative pixel coordinates will not effect the outcome of triangulation.

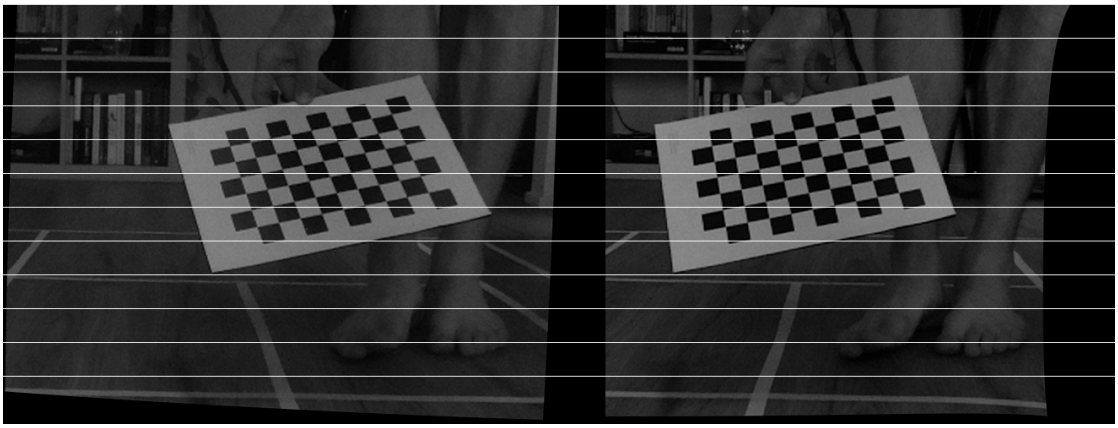

Figure 24: Rectified Left/Right Images

As described in Section 2.2, remapping the entire image is a time consuming process and is unnecessary if the relevant ball coordinates have already been obtained. Therefore the projection and rotation matrices are instead used in OpenCVs *undistort()* function to rectify individual points in the image.

7.2 Stereo Correspondence

There are two overarching methods for finding the correspondence between two images, correlation based and feature based. Correlation based aims to search for areas in both images that are similar, detecting things such as corners, edges and textures. Feature based methods rely on finding features in the image, then examining those features to determine the matching points between images.

Feature based stereo correspondence is the clear choice for this project, as a list of rectified ball candidates with object features has already been detected in each image, they can now simply be compared to corresponding features in the other image to determine point matches. To achieve this, all points in in the left and right images were iterated through, with points that lie within a range of y coordinates being passed into a similarity function, as ideally corresponding points lie on the same horizontal line after rectification due to the epipolar constraint.

The similarity function was designed to compare the ratio between the height (h_r), width (w_r) and size (s_r) of objects in each image. Ratios were used over absolute differences, as the ball changes size significantly throughout the recording and therefore relative size comparisons are more applicable.

Firstly, each ratio was evaluated with the larger value as the numerator, always resulting in a ratio greater than 1. Secondly, the similarity function, see Eq 21, was used to evaluate the overall similarity (S) between objects. The output of the function ranges from 0 to 1, with 1 indicating that objects are identical and 0 indicating that objects are significantly different. Each ratio is squared to increase the sensitivity of the function to differences in ratio. The objects that have the highest similarity between images above a pre-defined threshold are deemed to be matching objects and are subsequently triangulated to obtain 3D world coordinates.

$$S = \frac{3}{{s_r}^2 + {h_r}^2 + {w_r}^2} \qquad \{[s_r, h_r, w_r] \epsilon \mathbb{R}, 1 < [s_r, h_r, w_r]\} \tag{21}$$

There are some caveats to the use of this method for stereo correspondence. Firstly, all points in both images are being evaluated. This effectively means that the processing time of the function increases with the square of the number of detected objects. This can of course be overcome by more aggressive filtering during object detection, at the cost of true positives being successfully detected. Additionally, any real unwanted objects

that are detected will be subsequently triangulated. This is less of a problem, as outliers will be removed by the subsequent shift token transfer trajectory analysis stage.

7.3 Triangulation

As objects between the two images have now been detected, rectified and matched, triangulation can now be used to obtain the 3D coordinates in world space. This is achieved using the *triangulatePoints()* function in OpenCV, which uses the projection matrices $P1\&P2$ from stereo calibration and the points from each image to solve a set of linear equations to obtain the 3D homogeneous coordinates. These homogeneous coordinates can then be converted to Cartesian coordinates by dividing each element (X, Y, Z) by the last element in the vector (W).

The point is now represented in the coordinate system of the stereo cameras, with the origin at the principal point of the left camera. A rotation and translation matrix must then be applied to the point to account for camera tilt around the x axis and to translate the origin to the point between the two cameras. In this way, if the stereo cameras are placed on the baseline in the centre of the court, the coordinates obtained after triangulation will be in the coordinate system of the court. These matrices are constant, set by the height and downward tilt of the cameras.

7.4 Testing

To ensure all system components up until this point function as expected and to obtain a baseline for possible accuracy, a 3D simulation of a tennis shot was conducted using Blender. The simulation was calibrated as described in Section 4, using a chessboard pattern moved around the scene. Each camera was set to record at 90fps, 640x480 resolution and placed 2m either side of the baseline at 3.5m elevation and a tilted downward at 10° and inward at 6°. Using the simulation allowed for repeatable tests with a static background and also enabled the collection of 3D ball coordinates in each frame to use as a ground truth, something that is not possible in a real world implementation.

A simulation was conducted with a ball fired at 100 km/h at 8.1° elevation. After processing, the resulting 3D points were displayed on a 3D plot, see Figure 25. The ball detection was conducted using the same parameters previously identified in Section 6.4, resulting in a true positive rate of 72.6%.

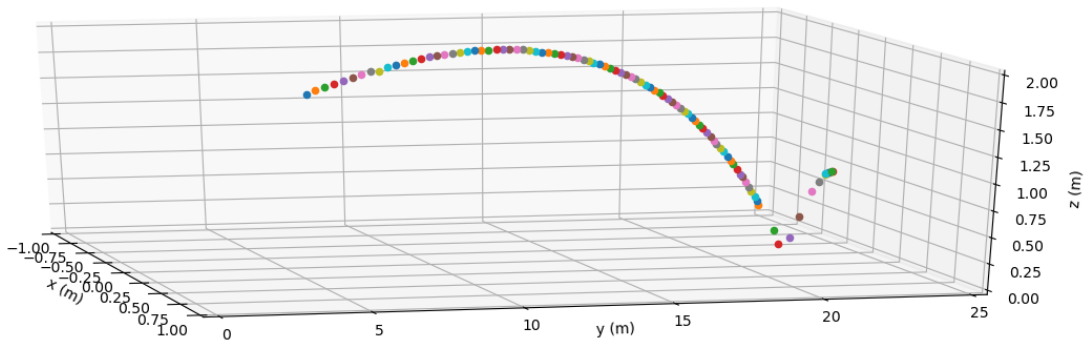

Figure 25: Triangulated Ball Coordinates (simulation)

The error (E) between ground truth (P_g) and detected 3D points (P_e) was calculated using euclidean distance in 3D space, see Eq 22.

$$E = \sqrt{(x_g - x_e)^2 + (y_g - y_e)^2 + (z_g - z_e)^2} \tag{22}$$

Root-mean-square error (RMSE) was then used to quantify the error over the entire set of points, see Eq 23. RMSE provides a real valued approximation of overall error by aggregating the magnitude of all errors over time into a single measure. It is also sensitive to outliers, making it a suitable metric for this analysis.

$$RMSE = \sqrt{\frac{\Sigma_{n=1}^{N} E_n^2}{N}} \tag{23}$$

Firstly, analysis of RMSE across all points revealed a constant error of (x=0.02m, y=0.61m, z=-0.05m). This error appears to show that the system achieves very poor accuracy, as an error of 0.6m is substantial in comparison to the size of the tennis ball. However, assuming it can be measured and remains constant, this error can simply be factored into subsequent calculations as an offset. Further analysis of the adjusted points reveals that the RMSE is approximately 0.04m across all points. This is an acceptably low overall error, given that it is approximately the diameter of a tennis ball, which is adequate for the purpose of this system.

As described in Section 7.3, the error due to the change in disparity increases with the square of the distance from the stereo cameras. To observe the effect of this in the experiment, a graph was constructed displaying the change in error vs the change in distance, see Figure 26. A line of best fit was calculated using least squares and overlayed on the graph, showing the overall relationship between distance and error.

Contrary to expectations, a linear fit was more appropriate for the measured points, indicating that there is another significant factor influencing the accuracy of ball triangulation which is inversely related to distance. This is likely due to the observation that there is more uncertainty about where the centre of the ball is when it is close to the cameras given that it occupies many more pixels in the image, whereas further away the ball occupies only a small number of pixels.

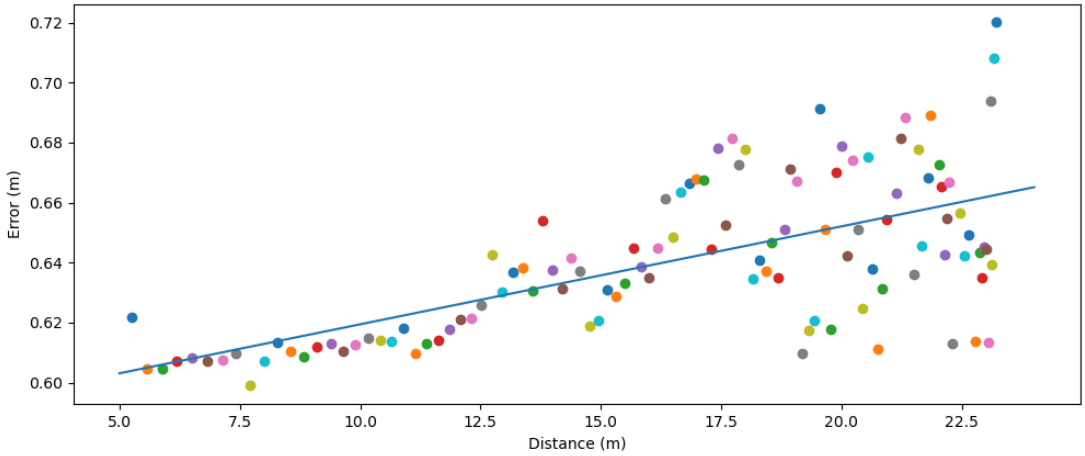

Figure 26: Euclidean error vs distance from stereo cameras (simulation)

8 Trajectory Analysis

To extract the most likely 3D ball trajectory from a set of 3D ball candidate points, a modified shift token transfer technique was utilised, similar to that shown in [11]. The aim of this technique is to connect the 3D ball candidates into tracklets based on their spatial and temporal relationship to each other. Once the tracklets are formed, they are also interconnected based on their spatial and temporal relationship and the tracklets are then analysed to determine the highest scoring combined tracklet which represents the most likely ball trajectory.

8.1 Formation of Tracklets

As tracklets become longer, there is more chance they will incorporate false ball candidates. To counter this, the range of frames from a recording [0,T] is broken into overlapping windows w_n of a fixed size. The set of ball candidates in each frame f_t are defined as $C_t = \{C_t^1, C_t^2 ... C_t^n\}$. The distance dM is used as the maximum distance the ball can travel in a single frame, which is defined by the frame rate and the maximum velocity of the ball. To predict the next position of the ball C_{t+1}^*, the first 3 ball candidates in each window that are within a 2dM radius are used to estimate the ball acceleration and velocity, using a constant acceleration ball motion model, see Eq 24.

$$\begin{aligned} Acc_t^k &= \frac{(C_t^k - C_{t-1}^k) - (C_{t-1}^k - C_{t-2}^k)}{\Delta T^2} \\ Vel_t^k &= \frac{C_t^k - C_{t-1}^k}{\Delta T} + Acc_t^k \Delta T \\ C_{t+1}^* &= C_t^k + Vel_t^k \Delta T + \frac{Acc_t^k \Delta T^2}{2} \end{aligned} \tag{24}$$

C_{t+1}^* is then compared to all C_{t+1}^k candidates, with each candidate given a score (S) using Eq 25, based on the deviation from the expected trajectory, which is the difference between the two vectors, see Figure 27.

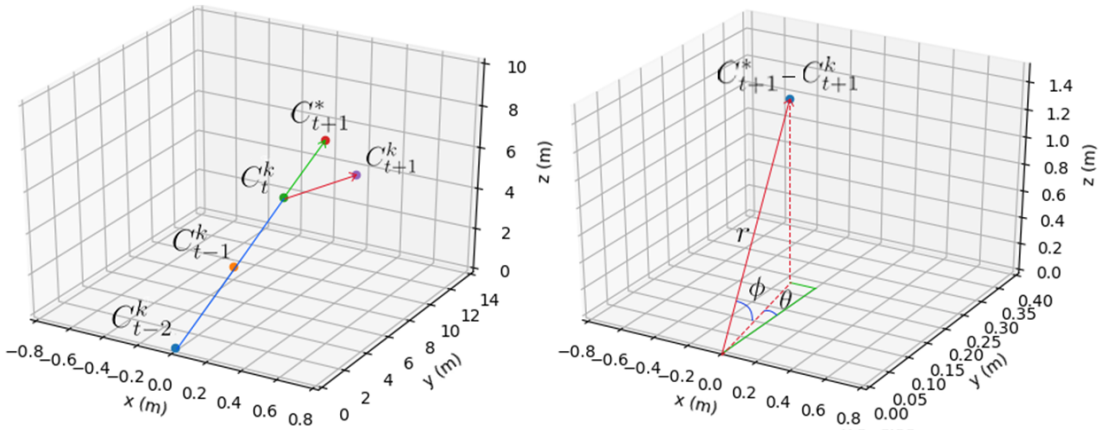

Figure 27: Candidate Estimation (left) and Difference Vector (right)

The difference vector is represented in spherical coordinates, with θ being the angle in the x-y plane, ϕ the angle from the x-y plane to the vector and r is length of the vector. A score for each dimension is calculated using an inverse exponential, which provides a score between 1 and 0, with the score decreasing exponentially as deviations increase.

$$\begin{aligned} S_1 &= e^{\frac{-r}{dM}} \\ S_2 &= e^{\frac{-\theta}{\pi}} \\ S_3 &= e^{\frac{-\phi}{\pi}} \\ S &= S_1 + S_2 + S_3 \end{aligned} \tag{25}$$

If S is above a score threshold, a token (tok^k_{t+1}) is added to the tracklet, storing the coordinates of the point and the score. This is repeated recursively until until there has been three repeated estimations with no valid candidates, upon which the tracklet is stored in an array of tracklets.

8.2 Combining Tracklets and Finding the Best Path

After all valid tracklets have been obtained, the tracklets are joined based on their temporal and spatial relationship to one another. To join the tracklets and find the overall highest scoring path, an acyclic graph based scheme is used. An acyclic graph is a directed graph with no cycles, meaning that each node in the graph may connect to another node, but there cannot be connections that create a loop back to that node,

creating a cycle. All valid tracklets are considered nodes in the graph, with the weight of each node being the cumulative score of all tokens in the corresponding tracklet.

There are two ways that tracklets can be combined, tip to tail overlaps and via intersection of extrapolated tokens. Tip to tail overlaps are primarily due to the overlapping of windows, see Figure 28. These can be identified by comparing the first N tokens of each tracklet t_n to the last N tokens of the previous tracklet t_{n-1}. If the tokens are sufficiently similar and consecutive, a link is added to the graph between t_{n-1} and t_n. Tracklets are then extrapolated from both the first and last token using the same constant acceleration ball motion model presented in Section 8.1, see Figure 28. If the extrapolated tracklet intersects another tracklet, within a specified margin of error, it is considered to be connected and a connection is made on the graph.

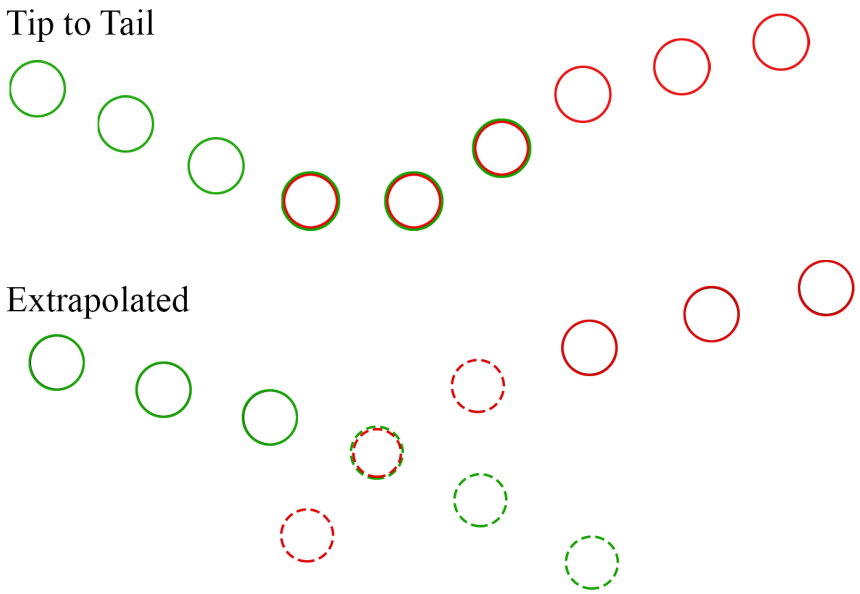

Figure 28: Methods for connecting tracklets

After the graph has been constructed, a list of start tracklets is defined by tracklets that are not on the secondary side of a connection from another tracklet and a list of end tracklets is defined by tracklets that have no outgoing connections. The longest path through the graph between each pair of start-end tracklets is then found using a modified solution to the single source shortest path SSSP problem for topologically sorted weighted directed acyclic graphs [21]. This solution takes linear time $O(n + m)$ on a graph with n nodes and m connections. Due to the nature of how the tracklets are constructed and merged moving from $t = 0$ to $t = T$, the tracklets are already

topologically sorted, with the score of each tracklet being the weight on the outgoing connections from that node.

A dynamic programming solution is used to implement this algorithm, operating recursively. Starting at the start node t_1 in Figure 29, the score of that node shown inside the circle is added to the overall path score. The algorithm then recursively examines each connected node t_k, adding the nodes score to the current path score and adding the node to the current path. If the node already has a path score, it has already been examined as part of another possible path. In this case, the lower of the two path scores is kept and the path at that node updated, working through the graph until reaching the end node.

The shortest path to each node in the graph has now been found, however to obtain the tracklet with the highest score, the longest path is required. This can easily be accomplished by negating all score values before finding the shortest path. Once the shortest path has been found (the most negative score in this case), it can be negated again to obtain the longest path with the highest score, shown by the green arrow tips in Figure 29.

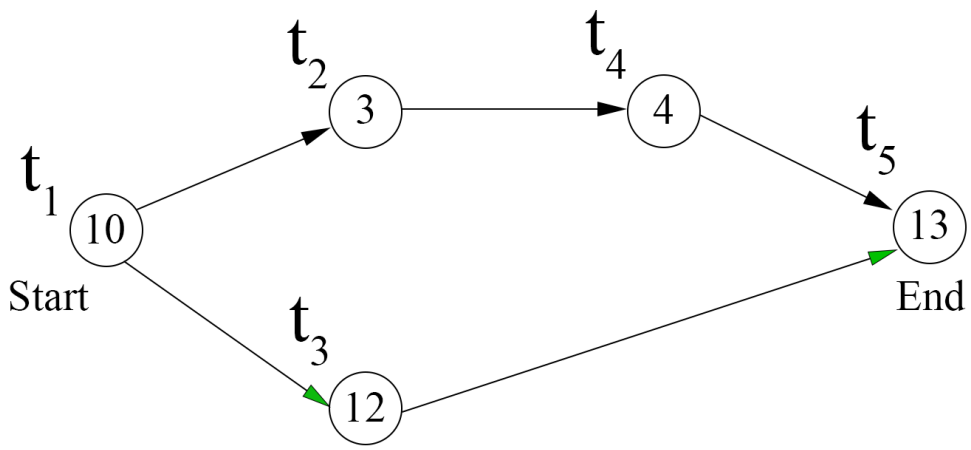

Figure 29: Directed Acyclic Graph with Longest Path

Once the longest tracklet has been found, it is segmented based on the local acceleration between consecutive tokens. If the local acceleration is negative, the ball is at a local minimum point (a bounce) and the tracklet is split between the two tokens. The first tracklet obtained from this split is assumed to be the desired trajectory from machine launch to first bounce on the court.

8.3 Curve Fitting

To model the entire ball trajectory to the first bounce of the ball on the court, a second order parametric equation, see Eq 26 is fit to each axis using a non-linear least squares fit.

$$\begin{aligned} x &= a_x + b_x t + c_x t^2 \\ y &= a_y + b_y t + c_y t^2 \\ z &= a_z + b_z t + c_z t^2 \end{aligned} \tag{26}$$

After the parameters for each equation have been found, they can be used estimate the exact point at which the ball strikes the ground. Additionally, the derivative of each equation can be used to estimate the ball velocity at any time t along the trajectory. Figure 30 shows the fitted parametric curve for a simulated tennis ball shot with all detected ball candidates. The point of impact was estimated to be (x=0.02m, y=22.32m, z=0m) at a velocity of 9.21m/s. Comparing this result to the ground truth impact point (x=0.02m, y=21.68m, z=0m) shows the estimated trajectory deviates significantly. However, once the constant error offset presented in Section 7.4 subtracted from the estimated bounce point, the point becomes (x=0.01m, y=21.79m, z=0m), which is within 0.11m of the ground truth bounce location.

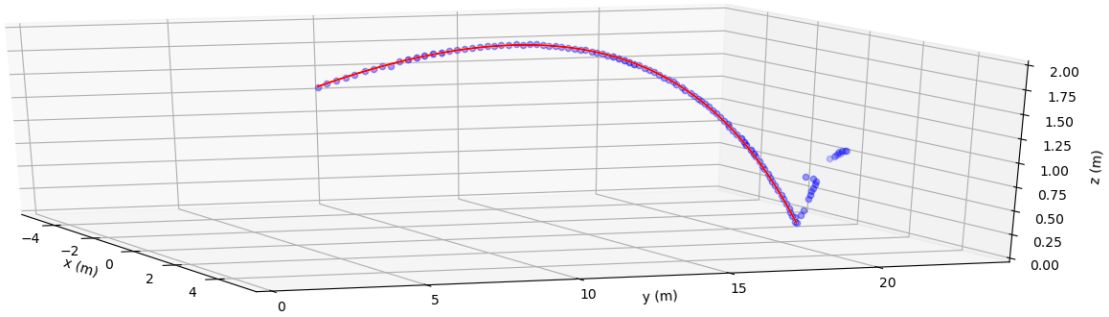

Figure 30: Simulation Trajectory - Parametric Curve (red) Ball Candidates (blue)

9 Small Scale Testing and Evaluation

9.1 1:10 Scale Model Overview

Due to the inherent constraints associated with full scale testing, including the stereo calibration issues presented in Section 4 and the limited access to tennis courts during the COVID-19 pandemic, a 1:10 scale model was used to establish the efficacy of the system, see Figure 31.

Figure 31: 1:10 Scale Tennis Court

The small scale model uses the prototype stereo camera rig with a 40cm baseline (4m scaled) with a 1:10 scale tennis court marked out using masking tape. A scale tennis ball was also made out of polymer clay that closely approximates the tennis ball colour and 6.6mm (6.6cm scaled) diameter, see Figure 32. The target velocity of the scale tennis ball is 3.6m/s (36.1m/s scaled), which can be easily achieved by lightly tossing the ball across the court by hand.

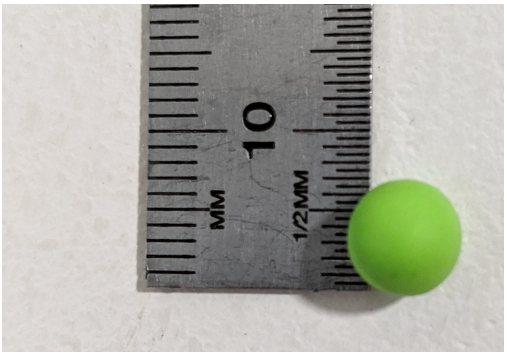

Figure 32: 1:10 Scale Tennis Ball

9.2 Testing and Results

A full system test was conducted to validate the overall functionality of the small scale prototype. The test environment is that shown in Figure 31, which is a well-lit indoor environment with a static but complex background. Both cameras and the server were connected to a local WIFI network hosted by the router mounted on the camera rig. The cameras were initialised and instructed to record for 2 seconds, with all image data, extracted foreground data and ball candidates saved locally for analysis. A scale tennis ball was thrown from approximately 20cm high in between the two cameras as the record command was send from the server to the cameras.

The total recording and analysis time, the time from the record command being issued to trajectory display was 2.5 seconds, successfully demonstrating the near real-time operation required by this system. The captured image data and foreground data was compressed along the time domain using image editing software to visualise the ball at captured points along its trajectory, see Figure 33.

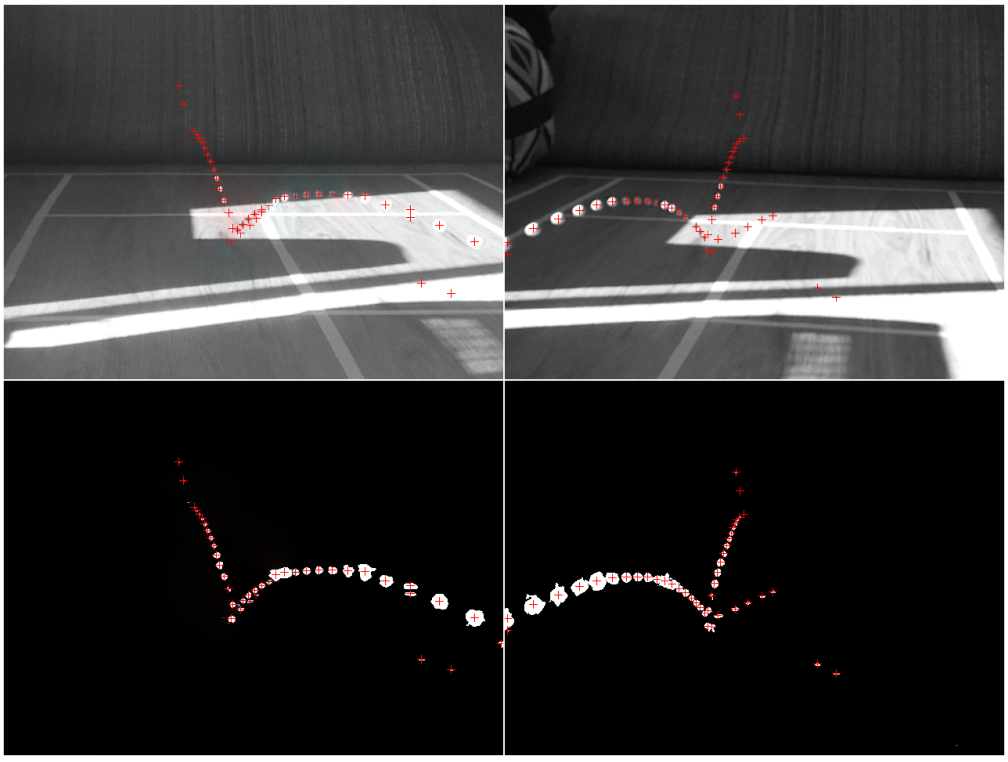

Figure 33: Left: Left Camera, Right: Right Camera, Top: Unprocessed Image, Bottom: Extracted Foreground. Red crosses indicate the centroids of ball candidates

The results from background subtraction and object detection confirm the outdoor experiment conducted in Section 6.4, as the vast majority of ball candidates are valid candidates with most false positives caused by the shadow or reflection of the ball.

The resulting triangulated ball candidates are shown in Figure 34. Although there are several outlying 3D points caused by the ball shadow, reflection or image noise, the ball trajectory and bounce location are clear to the observer. Although the ground truth ball location is not known in this experiment, the approximate location can be confirmed visually by comparing the bounce location in Figure 34 with Figure 33.

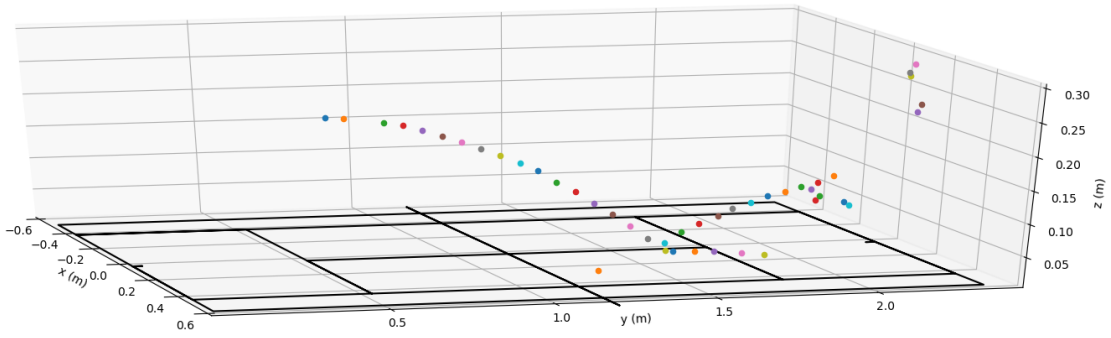

Figure 34: Triangulated Ball Candidates (1:10 scale)

The final ball trajectory was then estimated using the method described in Section 8, with the resulting trajectory shown in Figure 35. This trajectory is precisely that which the observer would assign to the ball from inspection of the sparse set of ball candidates shown in Figure 34. An estimate of the ball bounce location and velocity can also be obtained from the point at which the trajectory intersects the plane $z = 0$, found to be (x=-0.07m, y=1.71m, z=0m) at a velocity of 5.28 m/s.

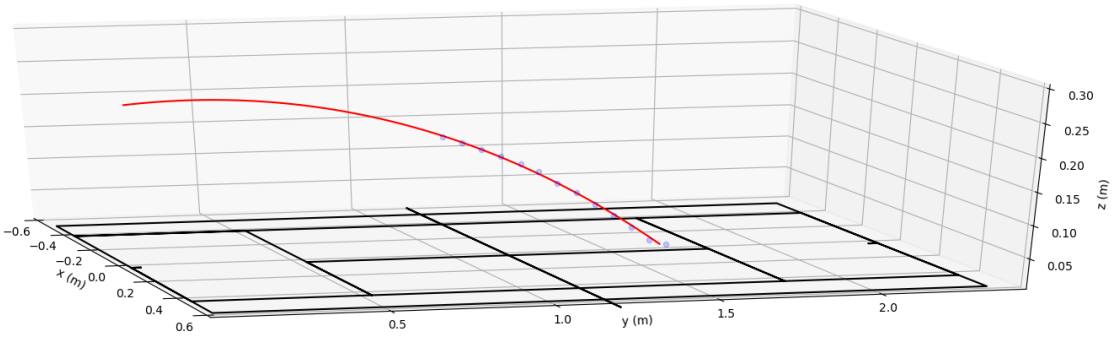

Figure 35: Final Ball Trajectory (1:10 scale)

10 Conclusions and Extensions

10.1 Conclusions

This research has successfully examined the current state of the art methods used for ball tracking in tennis. Due to the domain specifics of tennis, where a small ball moves at high velocity over a relatively large distance, robust, high speed techniques must be used to obtain near real-time performance of a tennis ball tracking system. To achieve near real-time processing, several computer vision techniques are combined with parallel processing and novel methods presented in relevant domain specific research.

A prototype low-cost solution to the problem is also presented, which can be implemented on widely available off the shelf consumer hardware using open source computer vision libraries. The system is capable of tracking a tennis ball travelling over 110 kph with an minimum full court error of 12cm and an average error of approximately 2cm across all points based on simulation results, assuming the constant error found in Section 7.4 can be accounted for in a real system. The prototype system has shown that near real-time analysis of tennis ball trajectories is possible using this method, with a 2s recording taking 2.5s to process and analyse the likely ball trajectory.

The prototype solution also demonstrates that relatively high accuracy can be achieved from a low-cost solution, as the total cost of the prototype hardware is approximately $320, see Table 2. This is ignoring the costs associated with mounting the cameras either statically or dynamically on a tennis court, as well as the cost of the power supply and network connection for each camera. Although these depend on the specific use case for the system, these would likely be under $180, keeping the total cost for the system under $500.

Table 2: Prototype Cost Breakdown

Item	**Cost ($AUD)**
Raspberry Pi 4 x 2	$ 223.80
Picamera v2 x 2	$ 77.90
3D Printed Mounts	$ 17.70
Total	$ 319.40

10.2 Extensions

There are a number of areas where this research can be extended to allow for this solution to work at full scale. Firstly, an alternative method for stereo calibration must be designed and tested, as discussed in Section 4. The current method for stereo calibration works well for cameras fixed relative to one another at small scale, but is overly cumbersome or a full scale system and is difficult to use for dynamic calibration. A stereo calibration method using rich features from the scene is likely to provide a sufficiently accurate stereo calibration in a full scale system.

Furthermore, to provide closed-loop feedback to the tennis trainer, the reference trajectory found in Section 2.5 and the measured trajectory found in Section 8 must be compared to generate an error signal for feedback to the machine. Designing this controller would require significant testing and validation on the real system, as there are at least five variables (velocity, spin, elevation, azimuth and wind speed) which must be accounted for.

A novel extension to this research would be to use the method presented to analyse tennis shots made by players and give feedback based on the trajectory characteristics. This would require an algorithm for player detection to be added to the background subtraction system and a trigger other than the tennis trainer for recordings, particularly if analysing a game of tennis rather than practice tennis shots.

References

[1] M. G. L. Bruce, 4th Baron Aberdare, and B. S. Lorge, *Tennis*, Encyclopædia Britannica, Sep. 5, 2019. [Online]. Available: https://www.britannica.com/sports/tennis/Organization-and-tournaments (visited on 09/23/2019).

[2] *Who is rené lacoste?* Lacoste. [Online]. Available: https://www.lacoste.com/gb/lacoste-brand/rene-lacoste.html (visited on 09/23/2019).

[3] *Phenom 2 specifications*, Lobster Sports. [Online]. Available: https://www.lobstersports.com/products/phenom2.htm (visited on 09/05/2019).

[4] T. King, "Tennis trainer," Final Year Engineering Project, University of Newcastle, 2019.

[5] *Hawk-eye in tennis*, Hawk Eye Innovations. [Online]. Available: https://www.hawkeyeinnovations.com/sports/tennis (visited on 09/09/2019).

[6] M. Polceanu, A.-O. Petac, H. B. Lebsir, B. Fiter, and C. Buche, "Real time tennis match tracking with low cost equipment," presented at the The Thirty-First International Florida Artificial Intelligence Research Society Conference, 2018. [Online]. Available: https://www.aaai.org/ocs/index.php/FLAIRS/FLAIRS18/paper/view/17690/16881.

[7] N. Owens, "Hawk-eye tennis system," in *International Conference on Visual Information Engineering (VIE 2003). Ideas, Applications, Experience*, IEE, 2003. DOI: 10.1049/cp:20030517.

[8] J. Mao, D. Mould, and S. Subramanian, "Background subtraction for realtime tracking of a tennis ball," presented at the International Conference on Computer Vision Theory and Applications, 2007. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.182.2043&rep=rep1&type=pdf (visited on 09/09/2019).

[9] T. D'Orazio, N. Ancona, G. Cicirelli, and M. Nitti, "A ball detection algorithm for real soccer image sequences," in *Object recognition supported by user interaction for service robots*, IEEE Comput. Soc. DOI: 10.1109/icpr.2002.1044654.

[10] F. Yan, W. Christmas, and J. Kittler, "A tennis ball tracking algorithm for automatic annotation of tennis match," in *Procedings of the British Machine Vision Conference 2005*, British Machine Vision Association, 2005. DOI: 10.5244/c.19.67.

[11] X. Zhou, L. Xie, Q. Huang, S. J. Cox, and Y. Zhang, "Tennis ball tracking using a two-layered data association approach," *IEEE Transactions on Multimedia*, vol. 17, no. 2, pp. 145–156, 2015. DOI: 10.1109/tmm.2014.2380914.

[12] A. Kaehler and G. Bradski, *Learning OpenCV3*. O'Reilly UK Ltd., Feb. 1, 2017, ch. 18, ISBN: 1491937998.

[13] K. Umeda, Y. Hashimoto, T. Nakanishi, K. Irie, and K. Terabayashi, "Subtraction stereo: A stereo camera system that focuses on moving regions," in *Three-Dimensional Imaging Metrology*, J. A. Beraldin, G. S. Cheok, M. McCarthy, and U. Neuschaefer-Rube, Eds., SPIE, 2009. DOI: 10.1117/12.805718.

[14] H. Brody, R. Cross, and C. Lindsey, *The Physics and Technology of Tennis*. USRSA, Apr. 1, 2004, ch. 42, p. 369, 450 pp., ISBN: 0972275908. [Online]. Available: https://www.ebook.de/de/product/3442945/howard_brody_rod_cross_crawford_lindsey_the_physics_and_technology_of_tennis.html.

[15] R. Obead, *Frame-accurate video synchronization using multi-raspberry pi camera module*, School Of Information Technology, Carleton University, 2018. [Online]. Available: https://www.academia.edu/38499974/Frame-Accurate_Video_Synchronization_Using_Multi-Raspberry_Pi_Camera_Module.docx (visited on 10/30/2019).

[16] A. Allan. (Jun. 24, 2019). "Benchmarking machine learning on the new raspberry pi 4, model b," [Online]. Available: https://blog.hackster.io/benchmarking-machine-learning-on-the-new-raspberry-pi-4-model-b-88db9304ce4.

[17] D. Jones, *Picamera - camera hardware*. [Online]. Available: https://picamera.readthedocs.io/en/release-1.13/fov.html#hardware-limits (visited on 10/30/2019).

[18] (2019). "Ball approval tests," International Tennis Foundation, [Online]. Available: https://www.itftennis.com/technical/balls/approval-tests.aspx (visited on 10/06/2019).

[19] (Sep. 27, 2019). "Multi-dimensional image processing (scipy.ndimage)," [Online]. Available: https://docs.scipy.org/doc/scipy/reference/ndimage.html (visited on 10/06/2019).

[20] *Digital camera noise*. [Online]. Available: https://www.cambridgeincolour.com/tutorials/image-noise.htm (visited on 03/21/2020).

[21] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. The MIT Press, Sep. 1, 2009, pp. 655–657, ISBN: 0-262-03384-4. [Online]. Available: https://www.ebook.de/de/product/8474892/thomas_h_cormen_charles_e_leiserson_ronald_l_rivest_clifford_stein_introduction_to_algorithms.html.

Appendix A - Time Allocation Breakdown

Table A-1 shows a breakdown of the project into major elements and the time taken to complete each item. Time taken is based on the project journal which was updated daily throughout the project.

Table A-1: Project Time Breakdown

Work	**Time (hrs)**
Research	76
Image Capture and Camera Calibration	78
Background Subtraction and Object Detection	56
Communication and Networking	57
Stereo Correspondence and Triangulation	20
Trajectory Analysis	47
Prototype Design and Construction	31
Debugging and Code Refactoring	50
Report Writing	81
Total	496

Appendix B - Tennis Ball Physics Model

Firstly, using the assumptions and equations presented in Section 2.5 and given the initial parameters for velocity v_0, elevation ϕ, azimuth θ, spin ω and wind conditions (v_w&θ_w), the components of the initial ball velocity can be found as follows.

$$\begin{aligned} v_x &= v_0 \cos(\phi) \sin(\theta) \\ v_y &= v_0 \cos(\phi) \cos(\theta) \\ v_z &= v_0 \sin(\phi) \end{aligned}$$

The ball velocity with respect to the air surrounding it can then be found.

$$v_x = v_x - v_w \sin(\theta_w) \qquad\qquad v_y = v_y - v_w \cos(\theta_w)$$

Using the above relations along with a free body diagram, see Figure 6, the following set of equations can be found by analysing the forces in each axis. Where: P is momentum, F is force, v is velocity and x is displacement.

$$\begin{aligned} \sum F_x = \frac{dP_x}{dt} &= -F_L \sin(\phi) \sin(\theta) - F_d \cos(\phi) sin(\theta) \\ &= -\frac{Adv^2}{2} \left(\frac{c_L v_z \sin(\theta)}{v} + \frac{c_d v_x}{v} \right) \\ &= -\frac{Adv}{2} \left(c_L v_z \sin(\theta) + c_d v_x \right) \end{aligned}$$

It should be noted that θ present in the lift coefficient part of the equation is a constant, as the spin axis is assumed to be constant from launch.

$$\begin{aligned} P_x &= m v_x \\ \frac{dx}{dt} &= \frac{P_x}{m} + v_{wx} \end{aligned}$$

This method is repeated to obtain equations for the y and z components.

$$\begin{aligned}
\sum F_y = \frac{dP_y}{dt} &= -F_L \sin(\phi)\cos(\theta) - F_d \cos(\phi) cos(\theta) \\
&= -\frac{Adv^2}{2}\left(\frac{c_L v_z \cos(\theta)}{v} + \frac{c_d v_y}{v}\right) \\
&= -\frac{Adv}{2}\left(c_L v_z \cos(\theta) + c_d v_y\right) \\
\frac{dy}{dt} &= \frac{P_y}{m} + v_{wy}
\end{aligned}$$

$$\begin{aligned}
\sum F_z = \frac{dP_z}{dt} &= F_L \cos(\phi) - F_d \sin(\phi) - F_g \\
&= \frac{Adv^2}{2}\left(\frac{c_L\sqrt{v_x^2 + v_y^2}}{v} - \frac{c_d v_z}{v}\right) - mg \\
&= \frac{Adv}{2}\left(c_L\sqrt{v_x^2 + v_y^2} - c_d v_z\right) - mg \\
\frac{dz}{dt} &= \frac{P_z}{m}
\end{aligned}$$

This set of coupled differential equations can then be solved numerically. For this purpose, an integration method from the Python Scipy module, *ODEINT()*, was used to solve the equations. Figure B-1 shows the effect of 10m/s of wind at 90° on balls launched at different azimuth angles.

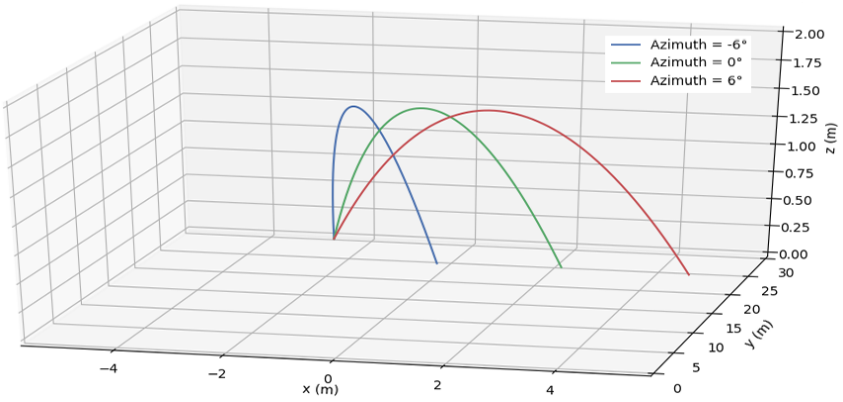

Figure B-1: Simulation Results wind 10m/s

Appendix C - Picamera Architecture

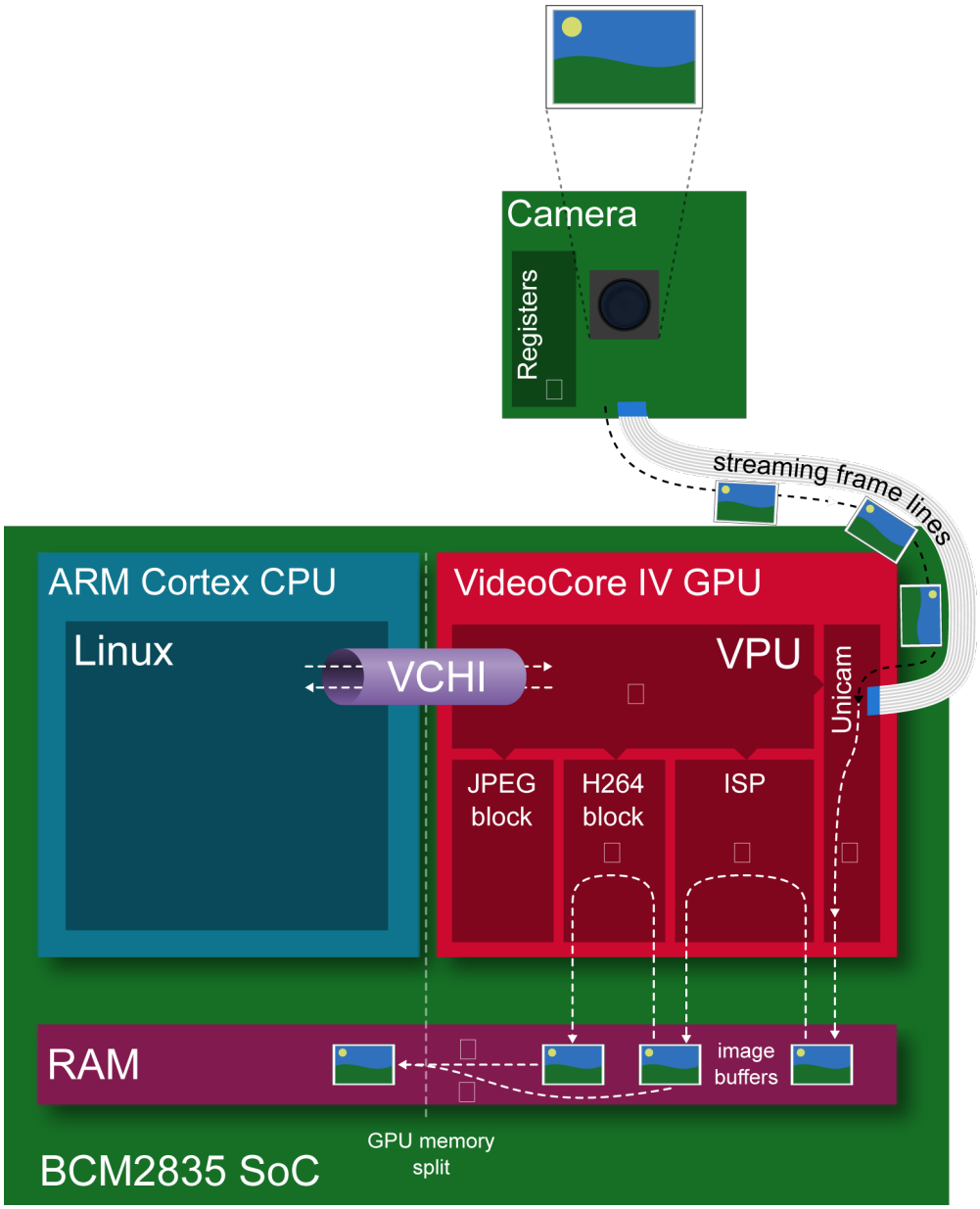

Figure C-1: Picamera Architecture [17]

Appendix D - Stereo Camera Court Coverage (Standard PiCam V2 AOV)

Figure D-1 shows the area of the tennis court covered by each camera, with the overlapping stereo region shaded in a different colour and dimensions in m.

As disparity resolution increases with the distance between cameras, it is desirable to place the cameras as far apart as possible, while keeping the cameras as frontal parallel as possible and maximising the overlapping area. An approximate camera baseline of 4m with cameras tilted inward at 6° was estimated graphically to balance these criteria.

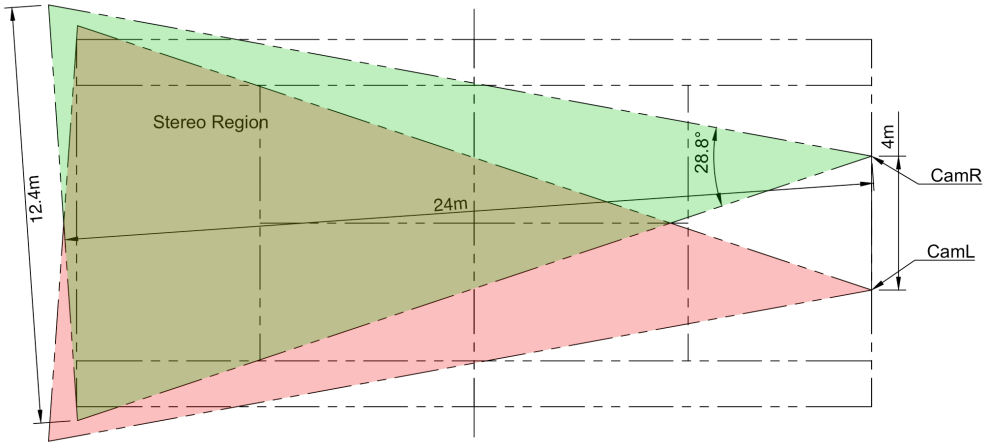

Figure D-1: Court Area Coverage