Faculty of Science and Technology

Department of Computer Science

UNDERGRADUATE INDIVIDUAL PROJECT

CST 3990 - Mobile Application that detects offside

Author: Hassan Yusuff **Student ID: M00919866**

Supervisor: David Gamez **Date: 25/04/2024.**

Table of Contents

ABSTRACT

This project's aim was to develop a mobile application to detect offside in football matches to provide an affordable solution for lower-tier and amateur leagues where advanced officiating technologies like VAR and SAOT are unavailable. Initially, the application used advanced machine learning methods including YOLO. However, these methods fell short in further analysis and efficiency, struggling with challenges like reliable player detection and accurate ball timing. Consequently, the project shifted to a rule-based method using Node.js and advanced image processing, including color quantization to distinguish players by uniform color and a grid-based framework for precise player and ball localization. This streamlined approach significantly eased the offside detection. This paper details the application's development, its technological advances, practical benefits, and potential enhancements, positioning it as a viable officiating tool in settings with limited resources.

DECLARATION

I, **Hassan Yusuff**, affirm that the work presented in this report and all related materials is entirely my own. I have properly cited all information derived from existing literature within the text and have included a comprehensive list of references. None of this dissertation's contents have been submitted for any other degree or diploma at this or any other institution.

Signature

Date

25/04/2024

ACKNOWLEDGEMENT

I would like to express my gratitude to my supervisor, **Dr. David Gamez**, for his invaluable guidance and support throughout this project. His expertise allowed me to explore this project in depth, which helped me acquire a deeper understanding of various machine learning algorithms and their applications. His approach to problem-solving, which emphasizes breaking down complex issues into manageable parts, has been particularly enlightening. I am also thankful for the unwavering support of my friends and family, whose encouragement was essential to my progress and success in this endeavor.

1. INTRODUCTION

1.1 PROBLEM DEFINITION

The significance of offside detection in football matches has become indispensable due to the high number of offside decisions made in every game, which typically averages around 50, according to the University of Michigan. The conventional method of on-field referees making manual decisions often results in contentious calls that can significantly influence the outcome of a match. The offside rule is complex, especially for casual fans. These complexities attract so many controversies from time to time during football games, leaving at least one team unsatisfied after a wrong call from the on-field officials. To tackle these challenges, the Video Assistant Referee (VAR) was introduced, but unfortunately, the VAR alone cannot solve the offside problem. Hence, the Semi-automated Offside Technology (SOAT) was introduced. The application of this technology effectively resolved the offside conundrum, but it also brought along a new difficulty: the prohibitive cost of integration. These advanced technologies, while enhancing the accuracy and fairness of decisions, come with high implementation and operational costs, making them inaccessible for smaller leagues and amateur competitions.

This economic barrier presents a significant challenge, particularly for five-a-side teams and lesser leagues that operate on much smaller budgets. These teams often lack the financial resources to invest in sophisticated technology like SOAT, which can cost millions of pounds for setup and maintenance. The lack of access to affordable offside detection technology not only puts these teams at a disadvantage during competitions but also affects the overall quality and fairness of the games they play. In response to this, the development of a mobile application that utilizes the principles of offside detection through accessible technology offers a promising solution.

1.2 MOTIVATION

This can be traced back to a personal experience during a casual five-a-side football match with friends. In this particular game, a dubious offside decision made by the referee significantly impacted the final outcome, contributing to my team's loss. The call was met with considerable dispute and discontent, casting a shadow over the fairness of the game, and leaving a sense of injustice among the players. This incident not only disrupted the harmony of the match but also highlighted a common problem in amateur sports. It was this exact moment that spurred a reflection on potential solutions to enhance the accuracy and fairness of officiating in sports, particularly in settings where professional-level technologies like VAR or SOAT are not viable options. This experience provided the initial motivation to develop a mobile application that could offer a cost-effective and accessible means to address these officiating challenges, thus promoting a more equitable playing field for all participants.

1.3 AIM

The goal of this project was to create a mobile application that simulates semi-automated offside technology using Machine Learning technologies such as YOLO (You Only look once) for Object Detection. The intention was to provide a cheaper-priced and more effective substitute for semi-automated offside technology, giving five-a-side football teams and lesser leagues an economical way to deal with offside detection. Although, considering the cost of the technology and availability of materials used to build it, the limitation of producing the same build of the semi-automated offside technology existed. However, the project initially faced significant challenges with machine learning approaches, such as accurately detecting team players.

To address these issues, the project shifted towards a rule-based analytical approach, integrating Node.js and sophisticated image processing techniques. This new strategy employs color quantization to distinguish players based on the RGB values of their uniforms. By using reference images, the system can accurately identify team colors, focusing exclusively on relevant pixel data. This approach significantly minimizes errors in player detection. The current results now process images through a structured grid-based framework. In this setup, each cell within the grid is classified by its dominant color, which corresponds to either a team player or the ball. This method ensures precise localization of players and the ball, which is crucial for correctly applying the offside rule during live matches. This paper details the development journey of the different approaches taken to achieve offside detection, emphasizing each technological advancement and practical implications. Future work will focus on enhancing the application's capabilities, including integrating it into a comprehensive system for live match scenarios. This will involve extensive field testing to ensure reliability and effectiveness in real-time football officiation.

2. LITERATURE REVIEW

2.1 BACKGROUND

Football is regarded as the most popular sport in the world, with an estimated Four billion fans spanning across all continents. It is a big industry and one of the biggest in the entertainment scene, countries like the USA and Canada call it soccer. Football has evolved from being merely a popular sport to an immensely wealthy and technologically advanced global phenomenon, using the 20-team English "Premier League" as an example, they make a healthy annual profit of about 40 million per team from television broadcast sales alone if you take away the revenue streams from jersey sales, ticket sales, and sponsorships.

The game's origins can be found in ancient China, more than 2,000 years ago, though the modern game as we know it began to take shape in England in the 19th century. Over these centuries, the game underwent significant transformations, with each era introducing new technologies and rules to enhance its appeal, fairness, and excitement. One of the pivotal moments in football's evolution was the formal introduction of the offside rule, which has been part of the game since its early codification in the 19th century. The offside rule was first instituted to stop players from unfairly placing themselves in front of the ball and close to the opponent's goal. The offside rule has been improved throughout time to maintain competition fairness while encouraging more dynamic play and goal scoring opportunities. However, the offside ruling is still one of the most disputed aspects of football today. Determining whether a player is offside entails determining whether they are ahead of the second-last opponent at the time the ball is passed to them, which can often occur in a matter of inches and milliseconds.

2.2 INTRODUCTION

Various advanced technologies have been introduced to tackle the offside issue, but they have not completely solved it. These technologies help professional and wealthy teams, leaving others without the same advantages. There is a need for affordable solutions that everyone in football can use, ensuring fair play and access to technology at all levels. In this literature review, I will review publications on Computer Vision used in sports, utilizing techniques like Simultaneous Localization and Mapping (SLAM), Object Detection Techniques such as YOLO (You Only Look Once), Faster Region-Based Convolutional Neural Network (Faster R-CNN). Additionally, publications explaining how technologies like the semi-automated offside technology aid referee decisions will be examined. This comprehensive review will explore how these advanced computer vision techniques and technologies contribute to enhancing the accuracy of decision-making in sports and similar competitive environments.

2.3 RELEVANT LITERATURES

2.3.1 Technological Advancements in football

Every aspect and industry in our world, including football, has been impacted by technology. A study by Bayraktar (2023) discusses major technological breakthroughs that have drastically changed the football environment while critically examining the transformative effect of technology in the sport. It spans a variety of technological innovations, including the Video Assistant Referee (VAR) System, Goal Line Technology, and Semi-Automated Offside

Technology (SAOT), all of which have revolutionized how games are officiated and watched. Bayraktar (2023) delineates these technologies and calls the VAR system an indispensable tool that helps referees make complex decisions through video replay technology.

The VAR system, a cornerstone of modern football refereeing, uses sophisticated multi-view video analysis technologies that enable referees to review judgments with unparalleled detail and accuracy. This technology uses innovative digital imagery and data analytics to scrutinize playbacks and make key decisions. Its success stems from its capacity to combine various camera viewpoints, offering a full view of situations that could otherwise go unnoticed in real time. This system aims to deliver quicker and more precise offside decisions, effectively reducing false detections and impulsive calls that can detrimentally impact the game's conditions. Hamdi et al. (2022) highlight how VAR improves accuracy and fairness, particularly in professional leagues. They presented a fresh compilation of football fouls taken from numerous camera perspectives in the paper and proposed a similar system called Video Assistant Referee System for Automated Soccer Decision Making from Multiple Views (VARS). Professional football officials painstakingly annotate these fouls, which serve as the foundation for approving the suggested VARS. It is trained to automatically identify the complex features of fouls as part of the benchmarking process, demonstrating how it can improve the accuracy of decisions made. In summary, the integration of video technology, shown by VAR and the innovative VARS plan, presents revolutionary opportunities for football referees. The suggested VARS system is a step in the direction of democratizing football decision-making and guarantees fair results at all playing levels. This evaluation contributes to the advancement of football refereeing methods by encouraging further research and development of video-assisted systems.

Another technological advancement in football is the Goal Line Technology. It is a ground-breaking instrument unveiled in 2014 that provides conclusive evidence of whether the ball has crossed the goal line in its entirety. It utilizes a combination of magnetic field sensors and high-speed camera systems. The magnetic sensors, installed within the ball and along the goal line, detect the exact moment the ball crosses the line, ensuring immediate and error-free confirmation. Bayraktar (2023) complemented this and explained how the deployment of 14 high-speed cameras around the stadium enables a 3D reconstruction of the ball's trajectory, offering clear, indisputable evidence for referees and audiences alike. This dual-technology approach guarantees decision accuracy in the most critical aspects of the game. Goal Line Technology solved issues like the ball crossing the line at an angle where cameras on the pitch could not get a clear image. But not the offside issue.

Bayraktar (2023) highlights how another advanced technology called Semi-Automatic Offside Technology (SAOT) aims to improve and speed up offside judgements by precisely determining the positions of players on the pitch by using up to 29 data points to make instantaneous offside calls, aided by a network of 12 specialized tracking cameras strategically placed around stadiums. Semi-Automatic Offside Technology (SAOT) represents a leap forward in addressing offside controversies, one of football's most contentious rules. It is stated that the system was created to work with VAR to improve officiating consistency, especially concerning offside calls. Bayraktar (2023) failed to address concerns such as the balance between technology and human judgment; this is presently a major issue in the football world. To successfully integrate modern technology, these complex obstacles must be professionally managed while maintaining the fundamental elements of the activity.

Despite these advancements in technology, integrating these sophisticated technologies into the conventional football environment still poses several

difficulties. Maintaining the delicate balance between human judgment and technological involvement, which is essential to the character and attractiveness of the sport, remains a source of concern. The effective implementation of these technologies requires a thorough evaluation of how they will affect the flow of the game, the authority of the referees, and the experience of the viewers. Furthermore, implementing these technologies will have a financial and logistical cost. As a result, creative solutions will be needed to maintain equity and accessibility throughout the sport, particularly for lower-tier leagues and clubs.

2.3.2 Computer Vision in sports

Recent advances in computer vision integration with sports analysis, particularly in football officiating, have sparked a revolution in the interpretation and administration of major events. A noteworthy stride in this approach is the work of Badami et al. (2019), which shows an innovative use of computer vision and image processing technology to address refereeing difficulties. This system possesses the capacity to decipher live video feeds and make instant decisions on crucial game events like goals, fouls, and offside situations. The core of this technology is its advanced player and ball tracking algorithms, which continuously assess players' and the ball's positions to give instantaneous input to officials on the pitch. This automation aims to enhance the consistency and precision of decisions in football matches by reducing the traditional reliance on human judgment. The system's foundation on computer vision—a field dedicated to enabling computers to extract and analyze information from visual inputs—heralds a new era in sports officiating. It entails complex procedures such as picture segmentation, object detection, and motion tracking, which help to accurately identify and track players and the ball in the dynamic surroundings of a football match. This technical advancement in precision and real-time analysis can drastically reduce the incidence of crucial decision errors, which

have historically plagued the sport.

Despite the promising outlook, Badami et al. (2019) research highlights several hurdles, including the smooth integration of such modern technologies into the diverse infrastructures of football stadiums. The report alluded to but did not go into detail about the technological and logistical challenges that must be overcome before this technology can be widely utilized. These include adapting existing stadium architectures to accommodate the necessary camera systems and computing hardware, as well as developing interoperable software platforms that can integrate with the diverse technologies already in use by football leagues and associations around the world. The financial ramifications of adopting such advanced technologies must be considered. High initial setup costs, combined with continuous maintenance charges, are significant impediments to acceptance, especially in smaller leagues and teams with little financial means. This economic dilemma highlights the necessity for scalable solutions that can adapt to the financial and infrastructural realities of various football organisations.

An early study by D'Orazio et al. (2009) explored the usage of multi-camera systems for automatic offside detection highlights the technical evolution of officiating in football. Their method uses synchronised camera feed processing to determine player and ball positions in real time by strategically deploying six stationary cameras around the pitch to reduce occlusion and perspective issues. This technology allows for a full multi-view analysis, which improves the accuracy of offside calls, a long-standing source of disagreement in the sport. The use of real-time photography from official matches for system validation, combined with quantitative accuracy checks against manually created ground truth data, demonstrates the potential for such technology to transform football officiating. These studies jointly illustrate the growing importance of modern technologies, such as computer

vision, in improving the fairness and uniformity of sports officiating. While they provide a peek into a future in which digital automation plays an important part in football, they also spark a broader conversation about the technological, logistical, and financial issues associated with the integration of such systems. Addressing these difficulties will be critical to realising technology's full potential to improve football, ensuring that developments benefit not only top-tier leagues but also smaller entities around the world.

2.3.3 The Current Offside Solution

In response to the need for a more in-depth examination of the technology behind football officiating, the current solution in the football world is the Semi-Automated Offside Technology (SAOT). This technology was used for the 2022 FIFA World Cup in Qatar, displaying its effectiveness in improving the precision and fairness of the game on football's grandest stage. This part of the literature review explores the complex technological architecture of SAOT and related advances that try to economically simulate its capabilities. The study by Chaterjee (2023) serves as the foundation for our discussion, demonstrating the use of artificial intelligence and optical tracking technologies to improve decision-making and justice in football. However, it falls short of fully exploring the technological complexities and issues that such systems present.

The core of SAOT and its envisioned cost-effective counterparts' rests on the constructive collaboration between AI and machine learning (ML) algorithms, powered by advanced computational frameworks like PyTorch, TensorFlow, and OpenCV. These platforms facilitate the real-time processing and analysis of high-volume data streams captured by optical tracking cameras, which carefully track the trajectory of the ball and each player's 29 distinct body points by employing an inertial sensor embedded in the ball. After being

analysed by deep learning models, this data allows the accurate detection of offside scenarios by examining the positions of players in relation to the ball and each other during play. A number of difficulties arise when combining different technologies into a coherent system, the main one among them being the synchronisation of hardware and software elements to guarantee the lowest possible latency and highest possible accuracy. The libraries provide comprehensive support for hardware acceleration, which boosts computational performance and is crucial to the building and optimisation of deep learning models for this purpose. In addition, these libraries offer high-level abstractions that make it easier to construct intricate machine learning algorithms, and their active communities provide priceless resources for debugging and iterative model improvement.

According to Chaterjee (2023), a major obstacle is the exorbitant cost of implementing SAOT, which is caused by the excessive cost of computing infrastructure and high-precision cameras. Using open-source technologies and community-driven development processes offers a feasible route towards the goal of a more approachable solution. Among these tools, OpenCV stands out as particularly important because of its extensive feature set designed for computer vision applications involving the processing of images and videos, as well as its easy interface with deep learning libraries. The effort to create a less expensive technology that simulates the task of SAOT needs a multidimensional strategy, concentrating not only on the technical development of AI models but also on the optimisation of hardware resources and the study of creative finance mechanisms to democratise access. This entails a continuous process of testing, validation, and modification aimed at striking a delicate balance between cost-efficiency and technological sophistication to ensure that such systems are widely applicable across various levels of football competitions.

In conclusion, while the technological spectacle of the 2022 World Cup

demonstrated SAOT's potential to revolutionise football officiating, the pursuit of more accessible alternatives necessitates a thorough understanding of the underlying technologies as well as a collaborative effort towards innovation. By addressing these issues, the professional and academic communities can contribute to the development of football into a more interesting, equitable game that is built on the values of justice and innovation in technology.

2.3.4 Exploring SLAM Techniques

The integration of Simultaneous Localization and Mapping (SLAM) technology into football officiating tools is a ground-breaking step towards improving the precision and reliability of important game judgements. SLAM, a vital technique originally developed for robots, provides a revolutionary way to achieve real-time environment mapping and localization, which is critical for autonomous navigation. This technology's application in sports, notably football, takes advantage of its ability to generate dynamic 3D maps of the playing field while correctly tracking the movement of players and balls. Makhubela et al. (2019) digs into the complexities of SLAM, explaining its dual role in creating precise maps of previously unknown locations while also tracking the sensor or camera's position within these places. This dual feature is especially useful in sports, where tracking the precise movement of several objects, e.g., players and ball, across a constantly changing environment (the pitch) is critical for fair and accurate officiating.

According to Makhubela et al. (2019), the move to Visual SLAM (V-SLAM) represents a significant advancement. V-SLAM uses vision sensors, such as monoculars and RGBD cameras, to navigate and map environments. This change to visual sensors creates new opportunities to capture precise characteristics of the game, including as player positions, ball direction, and

potentially disputed situations like offside positions or fouls, with unprecedented depth and accuracy.

However, the use of V-SLAM in football officiating is not without its challenges. Makhubela et al. (2019) mentioned many technological challenges, including variable lighting conditions, reflecting surfaces, and the necessity for precise loop closure detection to prevent drift throughout the mapping process. These issues demand powerful algorithms capable of managing the complex, dynamic scenes found in a football match.

Kazerouni et al. (2022) extend the possibilities of V-SLAM by investigating its integration with deep learning approaches, which can improve feature extraction and match. This is a vital component for precise player and ball tracking. This combination of V-SLAM and deep learning paves the way for the creation of more advanced officiating technologies capable of adapting to the fast-paced, unpredictable nature of football, guaranteeing that decisions are made on precise, dependable data. Despite these developments, there are still issues with SLAM technology's application in football. Significant infrastructure and technology support is needed due to the intricacy of precisely mapping a football pitch in real-time and the computational needs of processing enormous amounts of visual data. Furthermore, while integrating such technology into the current structure of football officiating, caution must be exercised to ensure that technology enhances, rather than overshadows, the human element of the game and the sport's traditional ideals. The exploration of SLAM and V-SLAM technologies in football officiating is resulting in a new era of precision and reliability in sports decision-making. By resolving the technical problems and employing these innovative tactics, football officiating can advance into the future and make the game more fair, exciting, and technologically advanced.

2.3.5 Exploring Object Detection Techniques

This section addresses the complexities of object detection technology, emphasising its core ideas, uses, and most recent developments that could revolutionise football analysis and officiating. A key component of computer vision is object detection, which uses techniques like YOLO (You Only Look Once) and the Region-Based Convolutional Neural Network (R-CNN) family, which includes Mask R-CNN and Faster R-CNN, to precisely locate and identify objects in images or video streams. These methods go beyond simple object recognition; they also entail intricate inference procedures, visualisation with labels and bounding boxes, and the ability to trigger actions particular to a given application depending on discovered things. This is particularly relevant in football, as recognising player positions and ball dynamics from camera feeds can have an impact on game fairness and outcome.

A detailed review by Zhong et al. (2019) emphasises the importance of object detection in sports, notably football, and puts it as a critical component of video analysis and picture understanding. The transition from handmade features to deep learning paradigms has resulted in a significant improvement in performance, allowing for the extraction of high-level semantic features while overcoming the limits of old methods. Similarly, Zhengxia Zao et al. (2023) provided a historical review of object detection evolution, highlighting deep learning's transformational influence in the field.

Souhail et al. (2014) investigated the intricacies and optimisations of employing cascade classifiers to detect many things concurrently, with a focus on practical applications. Because it addresses adapting object detection to different computing settings, including embedded platforms, which is essential for real-time analysis in sports, this research is especially pertinent to officiating football games. The study sheds light on the difficulties

in implementing real-time systems and emphasises the necessity of low-latency solutions, which are essential in hectic sporting contexts.

Additionally, Deshpande et al. (2020) investigates the developments in object detection algorithms, illuminating their practical uses in domains other than sports, such as IOT and AI. It is possible to understand the trade-offs between speed, accuracy, and processing demands by comparing algorithms such as R-CNN, ResNet, and YOLO. It is impossible to overestimate the importance of striking a balance between decision accuracy and real-time performance when officiating football, therefore these factors are crucial.

Lastly, the study by Abraham (2023) and the paper by Brownlee (2021) highlight the most recent advancements in YOLO, especially YOLOv8. These studies demonstrate how well YOLO performs in real-time object identification, striking a balance between speed and accuracy necessary for uses such as autonomous driving and, more importantly, sports refereeing. The development of YOLO highlights the continuous work to improve object localization, particularly for small or crowded objects. This problem is similar to situations in football games where precise player and ball placement are crucial.

Using these technological insights to officiate football promises to improve the game's precision and fairness while also creating opportunities for more sports technology research and development. The incorporation of object detection technology into football and other sports could change officiating paradigms and make the game more equitable and interesting for players, officials, and fans alike as it continues to advance. Thus, this investigation into object detection technology not only tackles the technological elements of football-officiating but also lays the groundwork for further developments in the area.

2.3 CONCLUSION

This comprehensive review explored the potential of employing contemporary artificial intelligence (AI) and machine learning (ML) methodologies to develop a system capable of detecting offside situations in football matches. Such development represents a highly promising opportunity for five-aside teams and lesser leagues by providing a low-cost alternative to the costly Semi-Automated Offside Technology, which is now only available to elite leagues and international federations. The papers assessed in this literature review provide compelling evidence of the continuous difficulty of offside identification and highlight the progress being made in computer vision research and its relevance to sports. Although it is recognised that some real-world applications might not reach industry standards due to resource limitations as well as the technological complexity of this project itself, the project's potential to provide a long-term fix for offside problems is still present. The development of a system that can detect offside automatically without requiring large spending would be a revolutionary step towards the democratisation and improved accessibility of sports technology. This kind of advancement not only holds the potential to improve the game at every level but also highlights how important AI and ML advancements are to closing the technical gap in the sports industry.

3. REQUIREMENT SPECIFICATIONS

3.1 Functional Requirements

Description	Priority
The app allows users to capture images using the device's camera	High
The app analyzes captured images for offside positions	High
The app notifies the user if an offside position is detected	High
Users can toggle between front and back cameras	Medium

3.2 Non-functional Requirements

Requirement	Specification
Usability	UI must be simple to use for non-technical users
Performance	Performance analysis of images should not exceed 5 seconds.
Compatibility	The app should be usable across iOS and Android devices.
Security	Images must be stored securely and comply with the relevant privacy regulations.
Maintainability	Code must be well-documented and structured for easy updates and maintenance.

3.3 Product

Name	Offside Detection app
Description	The Offside Detection app is a mobile application designed to capture images during a football game and analyze them in real-time to determine if players are in offside positions.
Features	• Real-time image capture and processing for offside detection. • User-friendly interface for initiating image capture. • Notification system for alerting the user of offside detections.
Primary Actor	User using device camera to take series of images
Basic Flow	A five-aside team wants to play ball, a person initiates the camera mode within the app to automatically capture images at set intervals by pressing the start button. The app processes these images in real-time to detect offside positions. If an offside is detected, the app sends an alert and

	stops further image capture, allowing for review and action on the detected incident.

4. PRODUCT

4.1 INTRODUCTION

Developing an application capable of detecting offside situation in football encompasses considerable complexity. The dynamic nature of football, characterized by rapid changes in player positions and ball movement, necessitates an algorithm that can accurately interpret these variables in real-time. Moreover, the precise identification of offside situations demands an intricate understanding of some key factors like the player’s position, ball's position, second-last defender's position, moment the ball is played, player's involvement in active play, player's position relative to the halfway line, direction of play, player receiving the ball directly from a goal kick, player's interference with an opponent, player movement speed and timing, and some more.

To address these challenges, my approach to creating such an application was to start a systematic investigation of different computational approaches to overcome these issues. Movement tracking, location analysis, and real-time data processing techniques were all carefully examined as part of this process, with each one chosen for its ability to provide a comprehensive solution.

In the sections that follow, I outline the series of creative approaches used, describing their potential strengths, advantages, and limitations in the context of offside detection. Hence, the ambition to engineer an application that consistently and accurately detects offside situations in football underscores the necessity for creative problem-solving strategies.

4.2 OBJECT DETECTION APPROACH

The goal of this project revolves around formulating a workable algorithm capable of detecting offside situations from a series of photos. The precise location of the players within these photos is a crucial component of this task as their spatial coordinates are essential for figuring out offside infractions. Object detection technology, therefore, emerges as a fundamental tool in this process, offering the means to automatically detect and position football players and the ball within digital imagery or video streams.

4.2.1 Selection of the Detection Model

Object detection in this case, can be achieved using a substantial number of technologies such as YOLO (You Only Look Once), SSD (Single Shot Detector), Faster R-CNN, Mask R-CNN, RetinaNet, EfficientDet, Anchor-Free Detectors (e.g., CornerNet, CenterNet). Identifying player and ball positions is vital for accurate offside rulings. YOLO stands out for its adoption here for its speed and efficiency which aligns with the fast-paced nature of football, its accuracy for ensuring reliable detection of players and the ball, and its single neural network that predicts classes and bounding boxes for the entire image in one evaluation, this is crucial for understanding player spatial relationships.

4.2.2 Data Acquisition

The foundation of this project was the collection of a large dataset from Roboflow, which was precisely built to capture a wide range of football match

circumstances. This collection is significant for its diversity, as it includes photographs taken from various angles, under varied lighting circumstances, and with participants at various phases of play. The precise annotations and labels in this dataset are necessary for effectively training object detection models. The dataset can be accessed here: https://universe.roboflow.com/smart-football-object-detection/smart-football-object-detection/dataset/11

4.2.3 Training Process and Model Refinement

Using the supplied detailed dataset, the project proceeded with the training of the YOLOv8 model. During this phase, the model's parameters were rigorously optimized to improve its detection performance in a variety of football game circumstances. The extensive training session, lasting over three hours, was of vital importance in tweaking the model's performance to precisely detect player positions and motions, which is essential to the purpose of this project.

Given the extensive requirements to run YOLOv8, which surpassed my laptop's capabilities, I encountered significant challenges with local installations, consistently facing errors and unsuccessful setups. Consequently, I turned to Google Colab's cloud services, opting for a cloud-based development environment to ensure seamless progress. This shift to cloud computing provided the necessary resources and stability for smooth development without the constraints of my hardware limitations.

1. **Set Environment**

```
In [ ]: from google.colab import drive
        drive.mount('/content/drive')

        Mounted at /content/drive

In [ ]: import os
        HOME = os.getcwd()
        print(HOME)

        /content
```

The steps involve:

- Importing the 'drive' module from the 'google.colab' library.
- Mounting Google Drive using the 'mount' function with '/content/drive' as the mount point.
- Importing the 'os' module.
- Retrieving the current working directory with the 'getcwd' function from the 'os' module.
- Printing the current working directory.

2. Create a Directory and Install Dependencies

```
In [ ]: !mkdir {HOME}/datasets
        %cd {HOME}/datasets

        !pip install ultralytics==8.0.134 opencv-python==4.8.0.74 roboflow --quiet

        /content/datasets
             ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 629.1/629.1 kB 6.5 MB/s eta 0:00:00
             ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61.7/61.7 MB 9.7 MB/s eta 0:00:00
             ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.7/58.7 kB 7.3 MB/s eta 0:00:00
             ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 155.3/155.3 kB 17.9 MB/s eta 0:00:00
             ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 178.7/178.7 kB 21.8 MB/s eta 0:00:00
             ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.8/58.8 kB 7.3 MB/s eta 0:00:00
             ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49.1/49.1 MB 13.5 MB/s eta 0:00:00
             ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 67.8/67.8 kB 8.9 MB/s eta 0:00:00
             ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 63.3/63.3 kB 8.3 MB/s eta 0:00:00
             ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 54.5/54.5 kB 6.9 MB/s eta 0:00:00
```

The steps involve:

- Use the `mkdir` command to create a directory called 'datasets' in the current working directory.
- Utilize the `%cd` magic command to change the current working directory to the 'datasets' directory you just created.

- Employ the `pip` package manager in quiet mode to install specific versions of libraries: 'ultralytics' version 8.0.134, 'opencv-python' version 4.8.0.74, and the 'roboflow' library.

3. Import Libraries and Perform Checks

```
]: from IPython.display import display, Image
   import ultralytics
   ultralytics.checks()

   Ultralytics YOLOv8.0.134 🚀 Python-3.10.12 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)
   Setup complete ✅ (2 CPUs, 12.7 GB RAM, 26.9/78.2 GB disk)
```

The steps involve:

- Importing the 'display' and 'Image' modules from the 'IPython.display' library.
- Importing the 'ultralytics' library.
- Executing the 'checks' function from the 'ultralytics' library to conduct essential validations.

4. Download Dataset from Roboflow

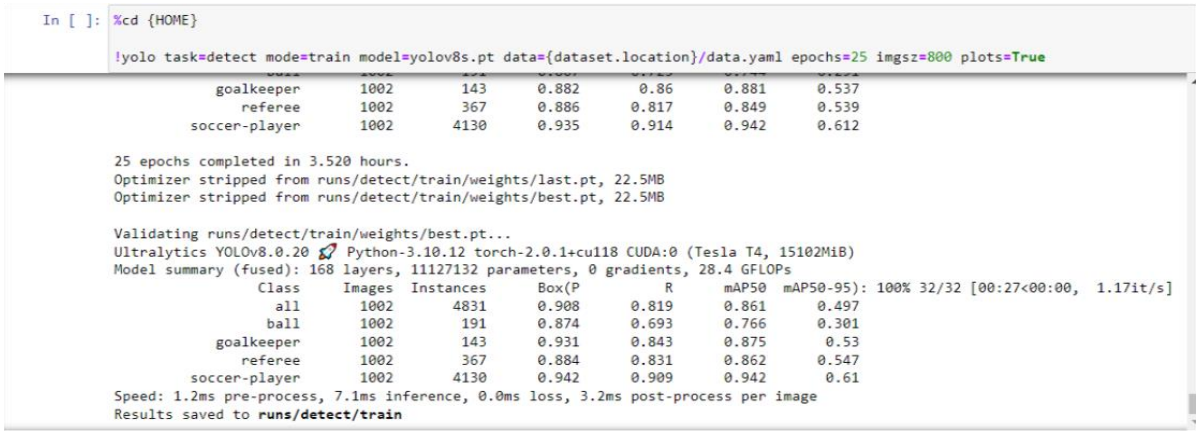
Dataset Link: https://universe.roboflow.com/smart-football-object-detection/smart-football-object-detection

```
]: from roboflow import Roboflow
   rf = Roboflow(api_key="***")
   project = rf.workspace("smart-football-object-detection").project("smart-football-object-detection")
   dataset = project.version(11).download("yolov8")

   loading Roboflow workspace...
   loading Roboflow project...

   Downloading Dataset Version Zip in Smart-Football:-Object-Detection-11 to yolov8:: 100%|██████████| 642312/642312 [00:19<00:00,
   33217.67it/s]
```

The steps involve:

- Import the `Roboflow` class from the `roboflow` library.
- Initialize a `Roboflow` class instance named `rf` with your API key.
- Use `rf` to access the "smart-football-object-detection" workspace, storing it in the `project` variable.
- Access the "smart-football-object-detection" project through `project`, assigning it to `dataset`.
- Download version 11 of the dataset, selecting "yolov8" as the model via the `download` method on `dataset`.

5. Perform YOLO Training



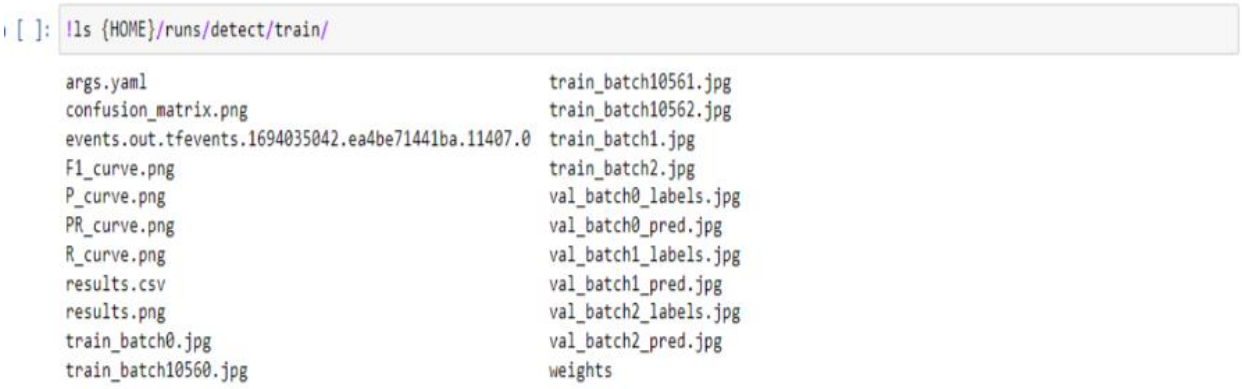
```
In [ ]: %cd {HOME}

        !yolo task=detect mode=train model=yolov8s.pt data={dataset.location}/data.yaml epochs=25 imgsz=800 plots=True

                  goalkeeper       1002        143      0.882       0.86      0.881      0.537
                     referee       1002        367      0.886      0.817      0.849      0.539
                soccer-player       1002       4130      0.935      0.914      0.942      0.612

        25 epochs completed in 3.520 hours.
        Optimizer stripped from runs/detect/train/weights/last.pt, 22.5MB
        Optimizer stripped from runs/detect/train/weights/best.pt, 22.5MB

        Validating runs/detect/train/weights/best.pt...
        Ultralytics YOLOv8.0.20 🚀 Python-3.10.12 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)
        Model summary (fused): 168 layers, 11127132 parameters, 0 gradients, 28.4 GFLOPs
                         Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 32/32 [00:27<00:00,  1.17it/s]
                           all       1002       4831      0.908      0.819      0.861      0.497
                          ball       1002        191      0.874      0.693      0.766      0.301
                  goalkeeper       1002        143      0.931      0.843      0.875       0.53
                     referee       1002        367      0.884      0.831      0.862      0.547
                soccer-player       1002       4130      0.942      0.909      0.942       0.61
        Speed: 1.2ms pre-process, 7.1ms inference, 0.0ms loss, 3.2ms post-process per image
        Results saved to runs/detect/train
```

The steps involve:

- Change the current working directory back to the original using the `%cd` magic command and the `HOME` variable.
- Execute the `yolo` command with arguments:
  - `task=detect` to specify detection as the task.
  - `mode=train` to set the mode to training.
  - `model=yolov8s.pt` to specify the use of the `yolov8s.pt` model.
  - `data={dataset.location}/data.yaml` to specify the dataset YAML file location.
  - `epochs=25` to set the number of training epochs to 25.

- `imgsz=800` to set the image size to 800.
- `plots=True` to enable the generation of training plots.

6. Visualize training results



The command `!ls {HOME}/runs/detect/train/` is utilized to list all files and directories located within the `{HOME}/runs/detect/train/` directory. This is typically performed to examine the results of the YOLO training process.

A breakdown of the command elements includes:

- `!ls`: The exclamation mark (`!`) precedes shell commands within Jupyter or Python notebooks, permitting their execution.
- `{HOME}/runs/detect/train/`: This path directs to the folder where YOLO training outputs are saved. The curly braces (`{}`) enable the integration of the `HOME` variable's value, indicating the home directory.

7. Validate Custom Model

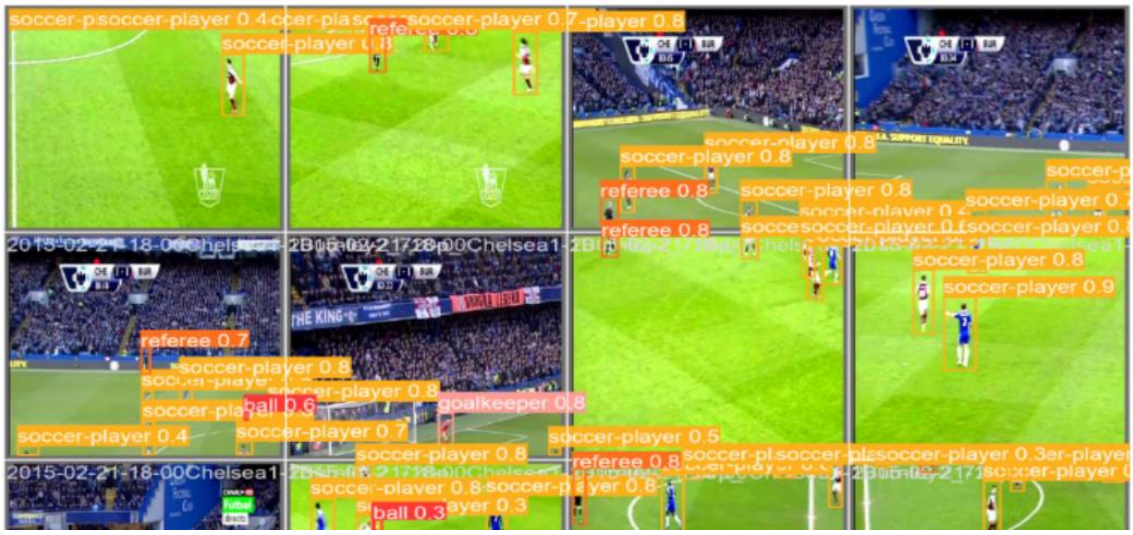
```
[ ]: %cd {HOME}

!yolo task=detect mode=val model={HOME}/runs/detect/train/weights/best.pt data={dataset.location}/data.yaml

/content
Ultralytics YOLOv8.0.134 🚀 Python-3.10.12 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 11127132 parameters, 0 gradients
Downloading https://ultralytics.com/assets/Arial.ttf to /root/.config/Ultralytics/Arial.ttf...
100% 755k/755k [00:00<00:00, 26.4MB/s]
val: Scanning /content/datasets/Smart-Football:-Object-Detection-11/valid/labels... 1002 images, 41 backgrounds, 0 corrupt: 10
0% 1002/1002 [00:00<00:00, 2274.37it/s]
val: New cache created: /content/datasets/Smart-Football:-Object-Detection-11/valid/labels.cache
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 63/63 [00:27<00:00,  2.27it/s]
                   all       1002       4831      0.909      0.817       0.86      0.496
                  ball       1002        191      0.869      0.686      0.754        0.3
            goalkeeper       1002        143      0.938      0.846      0.883      0.529
               referee       1002        367      0.886      0.828      0.861      0.547
         soccer-player       1002       4130      0.944      0.908      0.941      0.609
Speed: 1.0ms preprocess, 12.9ms inference, 0.0ms loss, 2.0ms postprocess per image
Results saved to runs/detect/val2
```

- `%cd {HOME}`: Utilizes the `%cd` magic command to change the directory to the one defined by the `HOME` variable. The `{}` syntax dynamically inserts the `HOME` variable's value.
- The `!` symbol is used to execute shell commands within the notebook. Here, it precedes the `yolo` command, which is followed by several arguments:
  - `task=detect`: Designates the operation to be carried out as object detection.
  - `mode=val`: Sets the operation mode to validation, indicating that the model will be evaluated on a validation dataset.
  - `model={HOME}/runs/detect/train/weights/best.pt`: Defines the path to the best-performing YOLO model weights. The `{}` syntax is used again to integrate the `HOME` variable's value dynamically.
  - `data={dataset.location}/data.yaml`: Specifies the path to the dataset's YAML configuration file, using `{}` to insert the `dataset.location` attribute dynamically.

When this code is executed, the YOLO model detects objects on the supplied validation dataset and produces bounding boxes and class labels for those items.

8. Examples of detections on the validation batch

[13]:



4.2.4 Technical Challenges with YOLOv8

YOLOv8 object detection algorithm was initially utilized which functioned well using the pre-trained data. However, it became evident that the pre-trained model is suitable for distinguishing between players from different teams, which is a crucial component of precise offside detection. Attempting to customize the training data to incorporate team-specific distinctions proved to be a complex and time-consuming task. Hence, retraining the model with a new, specialised dataset proved impractical due to the tight project deadlines. This resulted in the exploration of alternative image processing techniques better suited to address the project's unique challenges within the allotted period.

4.3 IMAGE PROCESSING APPROACH

After employing the machine learning approach without achieving the desired results, it became imperative to explore an alternative approach to fulfil the project's goals, especially considering the existence of a deadline. This section details the transition from the reliance on sophisticated machine learning models, to the implementation of a rule-based analytical method for offside detection in football. By utilising sophisticated image processing techniques along with the Node.js environment, this approach aims to detect offside positions from targeted coloured images. Applying colour quantization and spatial analysis within a structured grid-based framework simplifies the task of interpreting player positions and movements. The development of this offside detection system encompasses several stages, each critical to processing game imagery and accurately applying the offside rule. To summarize the entire approach and its flow; Initially, it computes average RGB colors from reference images of two teams and the ball using the `calculateAverageRGB` function. These colors are then dynamically mapped to specific labels ('team1', 'team2', 'ball') through the `setTargetColors` function. The main process involves reading game images, quantizing their colors for simplicity, and classifying each segment of a grid overlay on the image according to the proximity of its average color to the target colors. The system employs a depth-first search to detect contiguous regions in the grid, identifying players and the ball, and categorizes them accordingly. It further analyzes these detections to determine proximity to the ball and the tactical positioning of players. The offside decisions are made by comparing player positions across two sequential images to ascertain any offside occurrence when the ball is played. The system outputs the analysis results and manages errors throughout this process, leveraging advanced image processing.

4.3.1 Player Classification

In the development of this system, accurately identifying players dynamically from competing teams on the pitch is crucial. This necessitates the precise extraction of Red, Green, and Blue (RGB) color values associated with each team's uniforms. The process of accurately identifying the RGB values of team uniforms is crucial for the system's ability to effectively discern and classify players, which forms the basis for more complex analyses such as spatial positioning and offside determinations.

This targeted approach to player detection exemplifies an elegant balance between simplicity and functionality. Limiting the colour spectrum that the system analysed reduced the possibility of errors arising from the large variety of colours found in an average football game. As such, this initial stage of defining colour values is essential to the functioning of the system since it guarantees that only pertinent entities, such as players distinguished by the colour of their team and the ball, are taken into consideration throughout the offside detection procedure.

Code Explained:

```
import Jimp from "jimp";

async function calculateAverageRGB(imagePath) {
    try {
        const image = await Jimp.read(imagePath);
        let r = 0, g = 0, b = 0;
        let count = 0;

        image.scan(0, 0, image.bitmap.width, image.bitmap.height, function(x, y, idx) {
            // idx is the index of the pixel in the bitmap array
            r += this.bitmap.data[idx + 0];
            g += this.bitmap.data[idx + 1];
            b += this.bitmap.data[idx + 2];
            count++;
        });

        r = Math.round(r / count);
        g = Math.round(g / count);
        b = Math.round(b / count);

        console.log(`Average RGB: (${r}, ${g}, ${b})`);
    } catch (err) {
        console.error(err);
    }
}

calculateAverageRGB('./Data/images/scenario-red.png');
```

(property) Bitmap.data: Buffer

You, 1 second ago • Uncommitted changes

Figure 1: 4.3.1 Algorithm Explanation – Calculate RGB Value

A. **Importing Jimp:** The Jimp library is imported, providing the functionality needed for image processing tasks such as reading an image and accessing its pixel data.

B. **calculateAverageRGB function:** The calculateAverageRGB function is defined as asynchronous (async), indicating that it operates asynchronously and makes use of the await keyword to pause execution until the promise returned by Jimp.read is fulfilled

   I. **Reading the Image:** `Jimp.read(imagePath)` is called with the image path as an argument. This function asynchronously reads the image file from the given path and returns a `Promise` that resolves to a `Jimp` object representing the image.

   II. **Initializing Variables**: Before scanning the image, variables `r`, `g`, and `b` are initialized to `0`. These variables will accumulate the total values of the red, green, and blue colour channels, respectively, across all pixels. The `count` variable is initialized to `0` and will keep track of the total number of pixels processed.

   III. **Scanning the Image:** The `image.scan` method iterates over every pixel in the image. It starts from the top-left corner `(0, 0)` and covers the entire image area `(image.bitmap.width, image.bitmap.height)`. For each pixel, it executes a callback function that receives the current pixel's coordinates `(x, y)` and the index `idx` in the bitmap data array where the pixel's colour information starts.

   IV. **Accumulating Colour Values and Counting Pixels:** Inside the callback function, the RGB values of the current pixel are accessed using the `idx` and added to the running totals `r`, `g`,

and `b`. The `count` variable is incremented to keep track of the number of pixels processed.

V. **Calculating the Average RGB Values:** After scanning all pixels, the total sums of the RGB values are divided by the total pixel count to calculate the average values for each colour channel. The averages are rounded to the nearest integer using `Math.round`.

C. **Calling the Function:** Finally, the `calculateAverageRGB` function is called with the path to the image file as its argument, initiating the process described above.

D. **Result:**

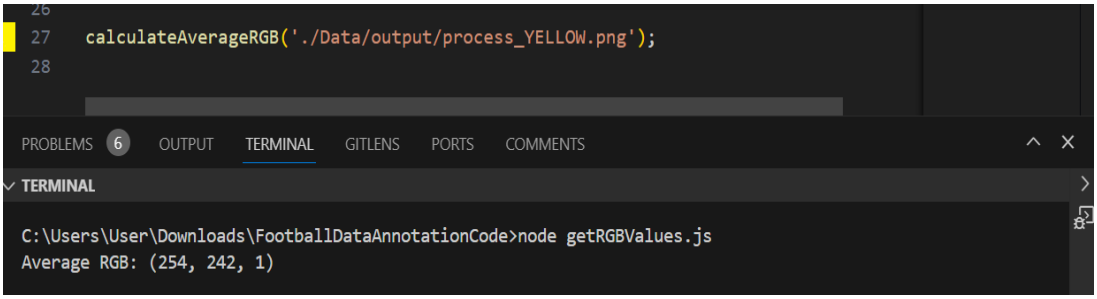


Figure 2: 4.3.1 Result – Calculate RGB Value

4.3.2 Image Processing and Grid Classification

The system begins with the segmentation of a game image into manageable cells, classifying each based on colour to identify distinct entities on the field:

1. **Reading and Parsing the Image:** Utilizes Node.js libraries to load and parse game imagery, preparing it for analysis.

```
// Import necessary modules
import { PNG } from 'pngjs';
import fs from 'fs';

// Define the sequence of image file paths
const imageFilePaths = [
    './Data/images/scenario11.png',
    './Data/images/scenario22.png'
];

// Asynchronously reads and parses an image file into a usable PNG object
async function readImage(filePath) {
    return new Promise((resolve, reject) => {
        fs.createReadStream(filePath)
            .pipe(new PNG())
            .on('parsed', function() {
                resolve(this);
            })
            .on('error', reject);
    });
}

// Simplifies the color space for comparison by quantizing color values
function quantizeColor(color) {
    // Reduce the color space by dividing the RGB values by 32 and rounding the result
    return { r: Math.round(color.r / 32), g: Math.round(color.g / 32), b: Math.round(color.b / 32) };
}
```

Figure 3: 4.3.2 Algorithm Explanation – Reading and Parsing the Image

2. **Calculating Average Cell Colour**: The `calculateCellAverageColour` function analyzes a specific rectangular segment (cell) of an image and computes the average colour of that segment. It does so by iterating over every pixel within the cell, summing up the Red, Green, and Blue (RGB) values of these pixels, and then dividing these total sums by the number of pixels to get the average RGB values. The result is a single RGB colour that represents the overall hue of the cell, simplifying the cell's colour information into a format that is easier to work with for image classification and analysis tasks. This process aids in identifying and classifying different entities in an image, such as players and the ball in a football match, by reducing the complexity of colour variations within each cell.

```
28
29  // Calculates the average color within a cell of the image
30  function calculateCellAverageColor(image, cellX, cellY, cellWidth, cellHeight) {
31      let rSum = 0, gSum = 0, bSum = 0, count = 0;
32      // Accumulate color values and count pixels within the specified cell
33      for (let dy = 0; dy < cellHeight; dy++) {
34          for (let dx = 0; dx < cellWidth; dx++) {
35              let idx = ((cellY + dy) * image.width + (cellX + dx)) << 2;
36              rSum += image.data[idx];
37              gSum += image.data[idx + 1];
38              bSum += image.data[idx + 2];
39              count++;
40          }
41      }
42      // Return the average color of the cell
        Test Regex...
43      return { r: rSum / count, g: gSum / count, b: bSum / count };
44  }
45
```

Figure 4: 4.3.2 Algorithm Explanation – Calculate Cell Average Colour Function

3. **Create Grid Classification**: The `createGridClassification` function classifies sections of an image into a grid based on predefined target colours, which represent different entities on the pitch, such as the two teams and the ball. This step is pivotal for preparing the image data for further analysis, such as identifying player positions and determining offside situations.

```
90  // Function to create a classification grid for the image based on target colors
91  function createGridClassification(image, squareSize, targetColors) {
92      const rows = Math.ceil(image.height / squareSize);
93      const cols = Math.ceil(image.width / squareSize);
94      const grid = Array.from({ length: rows }, () => Array.from({ length: cols }, () => "none"));
95
96
97      // Classify each cell in the grid based on the quantized color matches
98      for (let y = 0; y < image.height; y += squareSize) {
99          for (let x = 0; x < image.width; x += squareSize) {
100             const cellColor = calculateCellAverageColor(image, x, y, squareSize, squareSize);
101             const quantizedCellColor = quantizeColor(cellColor);
102             targetColors.forEach(targetColor => {
103                 if (quantizedCellColor.r === targetColor.r && quantizedCellColor.g === targetColor.g && quantizedCellColor.b === targetColo
104                     grid[Math.floor(y / squareSize)][Math.floor(x / squareSize)] = targetColor.label;
105                 }
106             });
107         }
108     }
109     // Return the classified grid
110     return grid;
111 }
```

Figure 5: 4.3.2 Algorithm Explanation – Create Grid Classification Function

a. **Determining Grid Dimensions:** The function first calculates how many rows and columns the grid will have, based on the image's dimensions and the square size. This step involves dividing the image's width and height by the square size and rounding it up to ensure the entire image is covered, even if it does not divide evenly into the grid size.

b. **Initializing the Grid:** An empty grid is initialized with dimensions calculated in the previous step. Each cell in this grid is initially set to "none," indicating that no classification has been made yet.

c. **Target Colors:** The function uses a list of target colours gotten from dynamically read images. Each colour is associated with a label (e.g., "team1," "team2," "ball"). These colors are intended to match the primary colors of the teams' uniforms and the ball. The colors are quantized to simplify the color space, making it easier to match colors in the image to these target colors.

d. **Classifying Cells**: The function then iterates over each cell in the grid. For each cell, it calculates the cell's average color using the `calculateCellAverageColor` function. This involves aggregating the RGB values of all pixels within the cell and computing their average, resulting in a single RGB color that represents the entire cell. The quantized version of this average color is compared against the list of target colors to find the closest match. This comparison is based on the quantized RGB values to account for minor color variations that might occur due to lighting conditions or material textures. Once the closest match is found, the corresponding label of the target color (e.g., "team1") is assigned to that cell in the grid.

4. **Result:** The outcome of the function is a simplified and classified grid that serves as a foundational dataset for further analysis. Imagine the football field as a digital grid created by the code. Each segment of the grid is analyzed to detect

specific colours corresponding to the players or the ball. When the system recognizes one of these targeted colours within a grid segment, it identifies which team the player belongs to, or if it is the ball. If no targeted colours are detected in a segment, it is simply marked as 'none'. This structured approach allows for precise identification and categorization of different elements on the pitch.

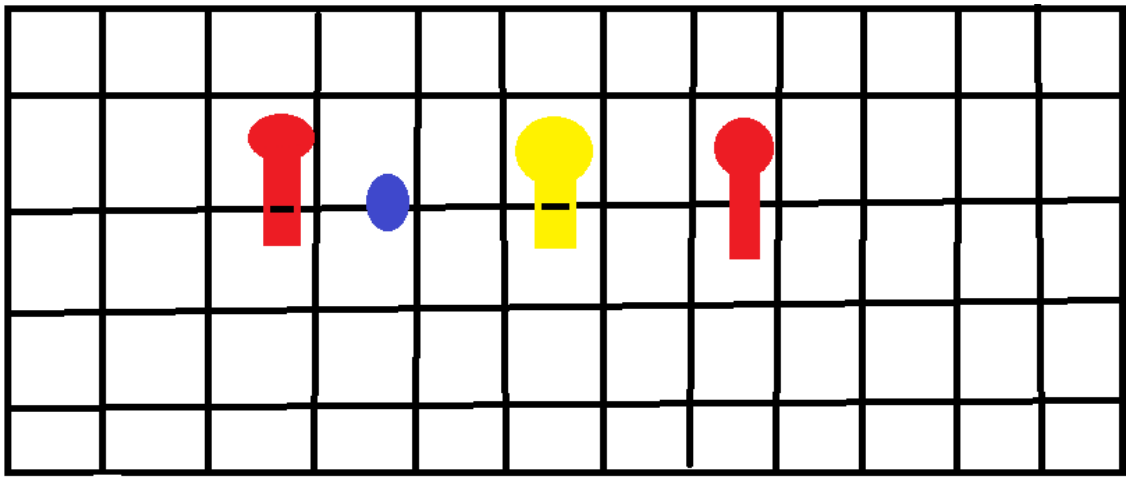


Figure 6: Example - Pitch Converted to Grid

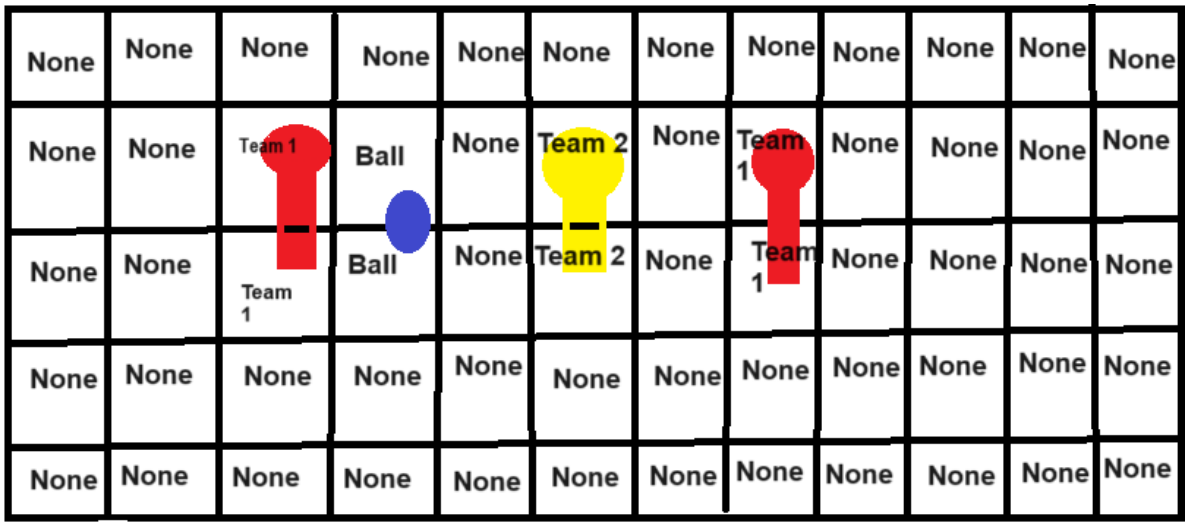


Figure 7: Example - Labelled Grids

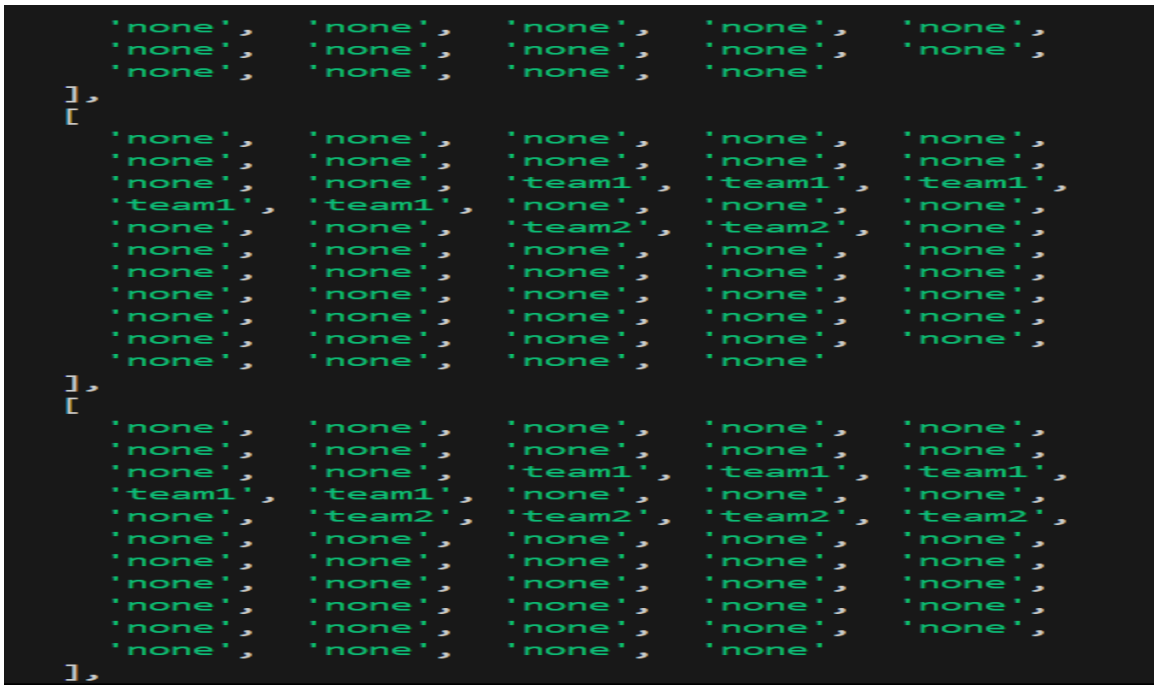


Figure 8: 4.3.2 Result - Console Result of Converted Grid

4.3.3 Detect Players and Objects

Detecting Players and Objects is a critical component of the automated offside detection system, designed to identify players and the ball on the football pitch from a segmented image grid. The `detectPlayersAndObjects` function was created for this purpose.

```
// Function to detect players, the ball, and calculate bounding boxes for each identified entity
function detectPlayersAndObjects(gridClassification, squareSize) {
    const visited = gridClassification.map(row => row.map(() => false));

    // Store detected players and objects
    const playersAndObjects = [];

    // Define directions for depth-first search
    const directions = [[1, 0], [0, 1], [-1, 0], [0, -1]];

    // Depth-first search to identify contiguous regions of the same label
    function dfs(x, y, label) {
        if (x < 0 || y < 0 || x >= gridClassification.length || y >= gridClassification[0].length || visited[x][y] ||
            return null; // Exit condition for recursion
        }

        visited[x][y] = true;
        // Initialize bounding box for the current region
        let bounds = { minX: x, minY: y, maxX: x, maxY: y };
        directions.forEach(([dx, dy]) => {
            const next = dfs(x + dx, y + dy, label);
            if (next) {
                // Update bounding box to include the adjacent cell
                bounds.minX = Math.min(bounds.minX, next.minX);
                bounds.minY = Math.min(bounds.minY, next.minY);
                bounds.maxX = Math.max(bounds.maxX, next.maxX);
                bounds.maxY = Math.max(bounds.maxY, next.maxY);
            }
        });
        return bounds;       You, 2 hours ago • Offside Logic fully implemented!
    }
```

Figure 9: 4.3.3 Detect Players and Objects Function – A

```
    // Iterate over the grid to detect and classify all entities
    for (let i = 0; i < gridClassification.length; i++) {
        for (let j = 0; j < gridClassification[i].length; j++) {
            if (!visited[i][j] && gridClassification[i][j] !== 'none') {
                const bounds = dfs(i, j, gridClassification[i][j]);
                if (bounds) {
                    // Create an object for the detected entity with its bounding box
                    playersAndObjects.push({
                        type: gridClassification[i][j],
                        x: bounds.minX * squareSize,
                        y: bounds.minY * squareSize,
                        width: (bounds.maxX - bounds.minX + 1) * squareSize,
                        height: (bounds.maxY - bounds.minY + 1) * squareSize
                    });
                }
            }
        }
    }
    // Return the list of detected players and objects
    return playersAndObjects;
}
```

Figure 10: 4.3.3 Detect Players and Objects Function – B

Here is a detailed explanation of how this function operates:

1. **Grid Classification:** Before invoking `detectPlayersAndObjects`, call the `gridClassification` function.

2. **Visited Array:** A two-dimensional array, `visited`, matching the dimensions of the `gridClassification`, is initialized to keep track of cells

that have already been processed. This ensures that the function does not process the same cell more than once, preventing infinite loops during the depth-first search (DFS) operation.

3. **Depth-First Search (DFS):**
    a. **Recursive Exploration:** The core of this function is a recursive depth-first search algorithm, implemented within the nested `dfs` function. DFS is a technique used for searching or traversing through the nodes in graph or tree data structures. Here, it is adapted to explore contiguous cells in the grid that share the same classification, indicating they are part of the same object (e.g., a player or the ball).

    b. **Directions Array:** The `directions` array defines the relative coordinates to move from one cell to adjacent cells in all four cardinal directions (up, down, left, right). This is used to explore all neighboring cells around a given cell.

    c. **Finding Contiguous Regions:** Starting from a cell not previously visited and with a specific classification (not 'none'), `dfs` explores in all directions, marking cells as visited and expanding the bounds of the detected object as it encounters contiguous cells of the same classification. This process effectively outlines the area occupied by an object (player or ball) on the grid.

4. **Bounds Determination:** As `dfs` explores and finds contiguous cells belonging to the same object, it calculates the minimum and maximum X and Y coordinates (`minX`, `minY`, `maxX`, `maxY`) of these cells. These coordinates define the object's bounding box, the smallest rectangle that encloses it.

5. **Object Recording:** Once the exploration of a contiguous region is complete, and the bounding box is determined, an object representing the player or ball is created and added to the `playersAndObjects` array. This object includes the type (classification), the top-left corner coordinates (`x`, `y`), and dimensions (`width`, `height`) of the bounding box. The dimensions are calculated based on the difference between the maximum and minimum coordinates, scaled by the `squareSize` to translate back to the original image dimensions.

6. **Grid Traversal:** The function iterates over every cell in the `gridClassification`. For each cell that has not been visited and is not classified as 'none', it initiates the DFS process to find and record the object it belongs to.

7. **Detected Entities:** The output of the function is the `playersAndObjects` array, which contains detailed information (type, position, and size) about each detected entity on the pitch. This array is essential for subsequent offside analysis.

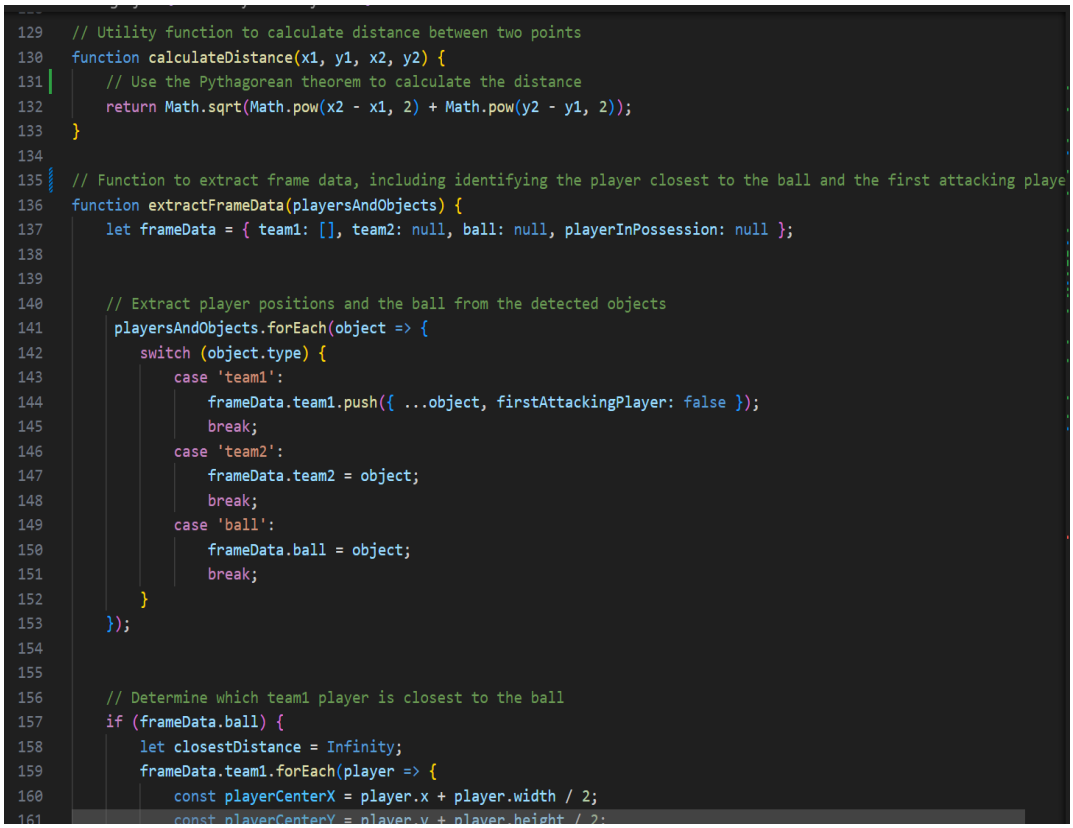
8. **Result:** The result of the function is a detailed, actionable dataset representing the players and the ball on the football field, extracted from the visual information of an image. This dataset includes the type of each entity (e.g., team1, team2, ball), along with its precise location (x, y coordinates) and size (width and height), enclosed within bounding boxes

```
Players and Objects Results:  [
  { type: 'team1', x: 120, y: 120, width: 130, height: 50 },
  { type: 'team2', x: 180, y: 210, width: 110, height: 40 },
  { type: 'ball', x: 250, y: 110, width: 20, height: 20 },
  { type: 'team1', x: 340, y: 260, width: 110, height: 30 }
]
```

Figure 11: 4.3.3 Result - Detected Players and Object

4.3.4 Frame Data Extraction

Following the detection of players and the ball, the next step is organizing that information into a structured format. This process is crucial for offering a clear visualization of the game's dynamics at the captured moment. It involves identifying the positions of players from both teams, locating the ball, and determining which player is in possession, all of which are pivotal for assessing potential offside situations. This is achieved using the `extractFrameData` function.



```
// Utility function to calculate distance between two points
function calculateDistance(x1, y1, x2, y2) {
    // Use the Pythagorean theorem to calculate the distance
    return Math.sqrt(Math.pow(x2 - x1, 2) + Math.pow(y2 - y1, 2));
}

// Function to extract frame data, including identifying the player closest to the ball and the first attacking playe
function extractFrameData(playersAndObjects) {
    let frameData = { team1: [], team2: null, ball: null, playerInPossession: null };


    // Extract player positions and the ball from the detected objects
     playersAndObjects.forEach(object => {
        switch (object.type) {
            case 'team1':
                frameData.team1.push({ ...object, firstAttackingPlayer: false });
                break;
            case 'team2':
                frameData.team2 = object;
                break;
            case 'ball':
                frameData.ball = object;
                break;
        }
    });


    // Determine which team1 player is closest to the ball
    if (frameData.ball) {
        let closestDistance = Infinity;
        frameData.team1.forEach(player => {
            const playerCenterX = player.x + player.width / 2;
            const playerCenterY = player.y + player.height / 2;
```

Figure 12: 4.3.4 Extract Frame Data Function - A

```
            const ballCenterX = frameData.ball.x + frameData.ball.width / 2;
            const ballCenterY = frameData.ball.y + frameData.ball.height / 2;

            const distance = calculateDistance(playerCenterX, playerCenterY, ballCenterX, ballCenterY);
            if (distance < closestDistance) {
                closestDistance = distance;
                frameData.playerInPossession = player;
            }
        });
    }

    // Logic to identify the first attacking player based on the highest height and width
    if (frameData.team1.length > 0) {
        let maxArea = 0;
        let indexMaxArea = -1;
        frameData.team1.forEach((player, index) => {
            let area = player.height * player.width;
            if (area > maxArea) {
                maxArea = area;
                indexMaxArea = index;
            }
        });

        // Mark the identified player as the first attacking player
        if (indexMaxArea !== -1) {
            frameData.team1[indexMaxArea].firstAttackingPlayer = true;
        }
    }
    return frameData;
}
```

Figure 13: 4.3.4 Extract Frame Data Function – B

Here is a detailed explanation of how this function operates:

1. **Player and Ball Extraction:** The function iterates over all detected objects, categorizing them based on their assigned labels ('team1', 'team2', 'ball'). Each player is added to their respective team array within the frameData object, with 'team1' players receiving an additional attribute to indicate if they are the primary attacking player. The ball's position is separately stored for easy access.

2. **Determining Possession:** To ascertain which player is in possession of the ball, the function calculates the distance between each 'team1' player and the ball. The player closest to the ball is presumed to be in possession, a key factor in offside decisions. This step underscores the importance of spatial analysis in understanding game dynamics.

3. **Identifying the First Attacking Player:** Among the 'team1' players, the one with the largest area (calculated as height multiplied by width) is designated the first attacking player. This heuristic assumes that players with a larger presence on the field are more likely to be in key attacking positions, an assumption that simplifies the identification process without delving into more complex motion analysis.

4. **Result:** The outcome of this function is a detailed overview of the image. It organizes player positions, the ball's location, and identifies important players. Again, this structured data is crucial for further analysis.

Figure 14: 4.3.4 Result - Extract Frame Data – A

Figure 15: 4.3.4 Result - Extract Frame Data – B

4.3.5 Offside Analysis and Decision Making

This is where the magic happens. It requires the execution of three crucial functions: 'processImageAndAnalyzeOffside', 'compareFramesForOffside', and 'analyzeOffsideScenarios'. This is the stage where each image from the specified file path is processed systematically to extract, arrange, and analyze game-related data. The subsequent step entails a detailed comparison of this organized data across multiple images to execute an offside analysis.

1. **Image Processing and Analysis:** The `processImageAndAnalyzeOffside` function is central to the application.

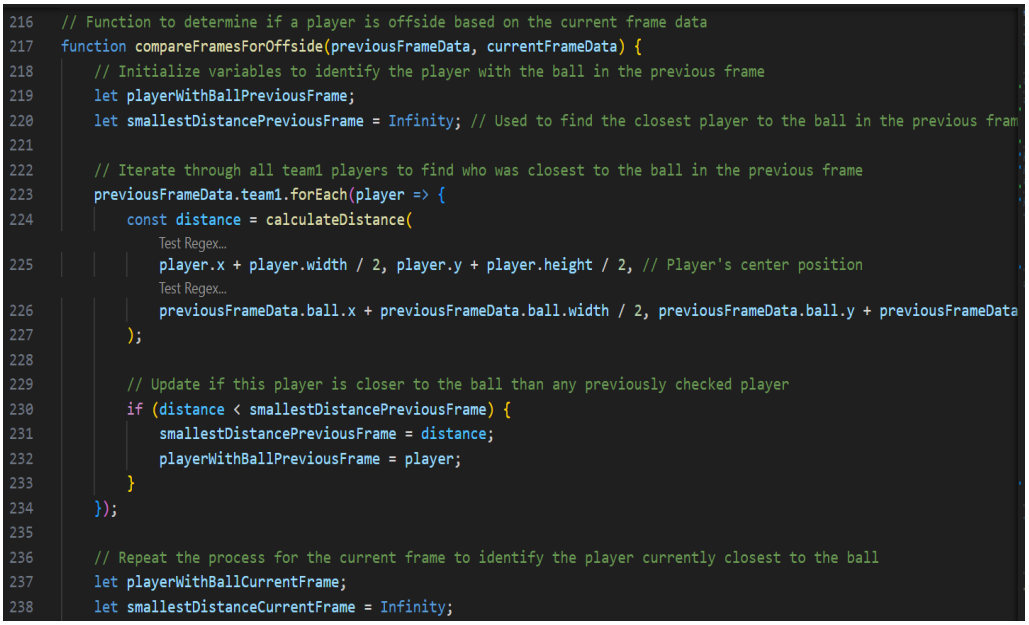
It starts by reading and parsing an image file into a format that can be analyzed. This step is crucial because it transforms the image into data that can be programmatically assessed. The next step involves creating a grid classification of the image. This process segments the image into a grid where each cell is classified based on predetermined target colors representing players from two teams and the football ball. This classification simplifies the complex visual data of a football match into a structured format that can be analyzed more effectively.

Once the grid classification is complete, the function proceeds to detect players and the ball within the grid. It identifies contiguous areas classified similarly (as either team players or the ball) and interprets these areas as distinct objects within the game. This identification is not trivial; it requires accurately distinguishing between different objects and classifying them accordingly. The final step in the process extracts meaningful frame data for offside analysis. It uses the previously identified objects (players and the ball) to determine their positions on the field, which players are in possession of the ball, and other critical pieces of information necessary for applying the offside rule.

```
async function processImageAndAnalyzeOffside(filePath, squareSize) {
    // Incorporating dynamic target colors into the grid classification function
    const targetColors = await setTargetColors();

    // Convert these colors using the quantizeColor function and continue with the rest of the processing
    const quantizedTargetColors = Object.values(targetColors).map(color => ({
        ...color,
        ...quantizeColor(color)
    }));

    // Read and parse the image.
    const image = await readImage(filePath);

    // Create a grid classification of the image.
    // This function divides the image into a grid and classifies each cell based on the target colors.
    const gridClassification = createGridClassification(image, squareSize, quantizedTargetColors);

    // Detect players and the ball based on the grid classification.
    // This function identifies contiguous areas of the same classification and interprets them as objects
    const playersAndObjects = detectPlayersAndObjects(gridClassification, squareSize);

    // Extract meaningful frame data for offside analysis.
    // Now, extractFrameData would organize this information, determining possession among other details.
    const frameData = extractFrameData(playersAndObjects);

    return frameData;
}
```

Figure 16: 4.3.5 Algorithm Explanation - Process and Analyze Image Function

2. **Offside Detection Logic:** The `compareFramesForOffside` function takes this structured data from two consecutive frames to determine if an offside condition exists between them. It first identifies which players were closest to the ball in both frames, marking them as the players in possession at those moments. By comparing the positions of these players and the ball across frames, along with the defender's position, the function applies the logic of the offside rule. If the player in possession changes between frames and the new player receiving the ball is ahead of the last defender, an offside is flagged. This simple yet effective logic mirrors the real-world application of the offside rule, considering the dynamic nature of football.



```
// Function to determine if a player is offside based on the current frame data
function compareFramesForOffside(previousFrameData, currentFrameData) {
    // Initialize variables to identify the player with the ball in the previous frame
    let playerWithBallPreviousFrame;
    let smallestDistancePreviousFrame = Infinity; // Used to find the closest player to the ball in the previous fram

    // Iterate through all team1 players to find who was closest to the ball in the previous frame
    previousFrameData.team1.forEach(player => {
        const distance = calculateDistance(
            player.x + player.width / 2, player.y + player.height / 2, // Player's center position
            previousFrameData.ball.x + previousFrameData.ball.width / 2, previousFrameData.ball.y + previousFrameData
        );

        // Update if this player is closer to the ball than any previously checked player
        if (distance < smallestDistancePreviousFrame) {
            smallestDistancePreviousFrame = distance;
            playerWithBallPreviousFrame = player;
        }
    });

    // Repeat the process for the current frame to identify the player currently closest to the ball
    let playerWithBallCurrentFrame;
    let smallestDistanceCurrentFrame = Infinity;
```

Figure 17: 4.3.5 Algorithm Explanation - Compare Frames for Offside Logic – A

```
    currentFrameData.team1.forEach(player => {
        const distance = calculateDistance(
            player.x + player.width / 2, player.y + player.height / 2, // Player's center position
            currentFrameData.ball.x + currentFrameData.ball.width / 2, currentFrameData.ball.y + currentFrameData.bal
        );

        // Update if this player is closer to the ball than any previously checked player
        if (distance < smallestDistanceCurrentFrame) {
            smallestDistanceCurrentFrame = distance;
            playerWithBallCurrentFrame = player;
        }
    });

    // Calculate the x-position of the team2 (defender) player
    const defenderX = currentFrameData.team2.x + currentFrameData.team2.width / 2;

    // Extract the X positions for relevant entities.
    const previousPlayerX = playerWithBallPreviousFrame.x + playerWithBallPreviousFrame.width / 2;
    const currentPlayerX = playerWithBallCurrentFrame.x + playerWithBallCurrentFrame.width / 2;
    const ballXAtPass = previousFrameData.ball.x + previousFrameData.ball.width / 2; // Position of the ball in the p
    const ballXAfterPass = currentFrameData.ball.x + currentFrameData.ball.width / 2;
```

Figure 18: 4.3.5 Algorithm Explanation - Compare Frames for Offside Logic – B

```
    // Identify if the player in possession is the first attacking player in each frame
    const playerInPossessionPreviousFrame =  previousFrameData.playerInPossession.firstAttackingPlayer;
    const playerInPossessionCurrentFrame =  currentFrameData.playerInPossession.firstAttackingPlayer;


    // If the same player retains possession from the previous frame to the current frame.
    if (playerInPossessionPreviousFrame === playerInPossessionCurrentFrame) {
        console.log("Same player retains the possession from the previous frame")
        // Not offside if the player was already in possession and crossed the defender with the ball.
        if (previousPlayerX <= defenderX || currentPlayerX <= defenderX) {
            return false; // The player is either behind or in line with the last defender, thus not offside.
        }
    } else {
        // If there's a new player in possession, indicating a pass.
        console.log("New player has possession in current frame")
        // It's offside if the second attacking player receives the ball and is ahead of the last defender
        if (previousPlayerX <= ballXAtPass && currentPlayerX >= ballXAtPass && currentPlayerX <= ballXAfterPass) {
            return true;
        }
    }

    return false; // Defaults to not offside if none of the specific conditions are met.
}
```

Figure 19: 4.3.5 Algorithm Explanation - Compare Frames for Offside Logic – C

3. **Analyzing Offside Scenarios:** Finally, `analyzeOffsideScenarios` orchestrates the analysis across a series of images. It ensures that the process is applied to exactly two images - representing two moments in a match - to establish whether an offside occurred during the transition. This function encapsulates the end-to-end process of offside detection, from reading the images and extracting the relevant data to applying the offside detection logic and presenting the results. The outcome of this analysis provides insights into the positions and movements of players relative to each other and the ball, offering a detailed examination of potential offside situations in football matches. This approach demonstrates the potential of image processing and analysis in understanding and interpreting sports rules and events, offering a new perspective on the application of technology in sports analytics.

```
async function analyzeOffsideScenarios(imageFilePaths, squareSize = 10) {
    if (imageFilePaths.length != 2) {
        console.error("This analysis requires exactly two images.");
        return;
    }

    // Process the first image to establish the baseline positions
    const initialFrameData = await processImageAndAnalyzeOffside(imageFilePaths[0], squareSize);
    console.log("Frame 1 result: ", initialFrameData)

    // Process the second image for offside analysis
    const finalFrameData = await processImageAndAnalyzeOffside(imageFilePaths[1], squareSize);
    console.log("\n\nFrame 2 result: ", finalFrameData);

    // Now, determine offside based on the comparison between the initial and final frames
    const isOffside = compareFramesForOffside(initialFrameData, finalFrameData);
    console.log(`Offside detected in the transition from the first to the second image: ${isOffside}`);
}



analyzeOffsideScenarios(imageFilePaths)
    .then(() => console.log("Completed offside analysis."))
    .catch(error => console.error("An error occurred during analysis:", error));

```

Figure 20: 4.3.5 Algorithm Explanation - Analyze Offside Scenarios

4.3.6 Image Processing Flow

The image processing algorithm for offside detection begins by loading images and translating them into quantized colour spaces to facilitate analysis. It then divides each image into a grid, allocating cells based on colour similarities associated with various team outfits and the ball. This grid is useful for identifying and categorising players and items on the pitch. By analysing these classifications, the algorithm determines player positions and interactions, which it then uses to identify probable offside situations based on changes between sequential frames. Here is a simplified diagram illustrating the workflow of the image processing algorithm:

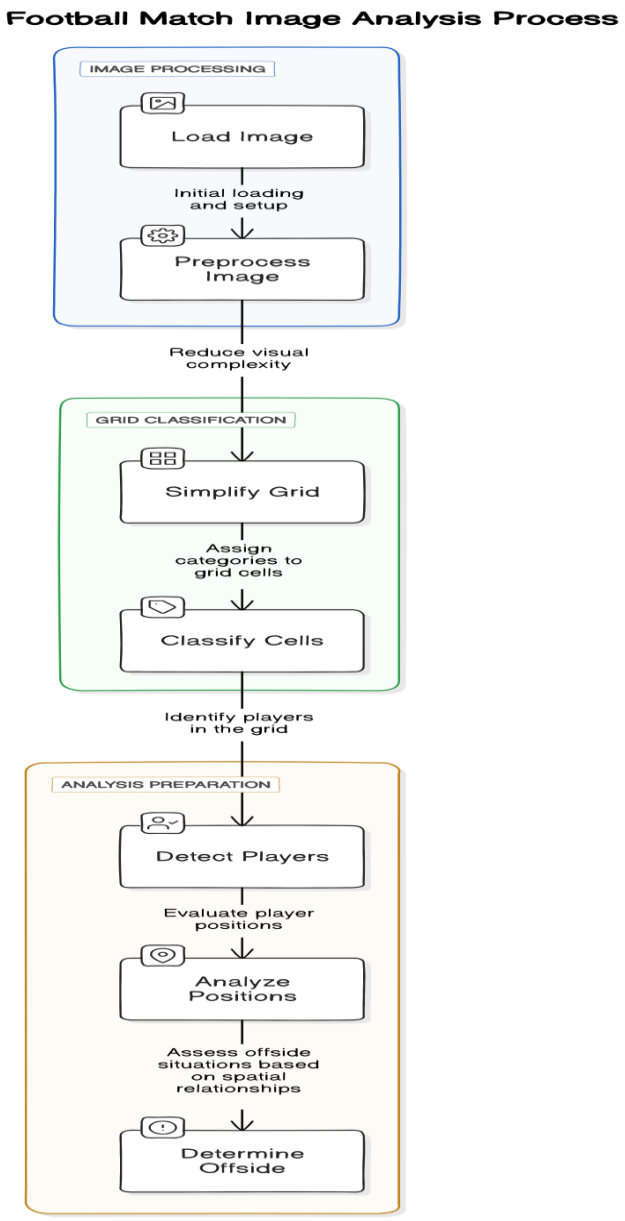


Figure 21: Image Processing Flow

4.3.7 Technical Challenges

The image processing approach to offside detection in football, though promising, faces several technical challenges primarily due to the highly

dynamic environment of live sports. Key among these challenges is the system's sensitivity to variable lighting conditions, weather changes, and diverse camera angles, which can significantly impact the quality and consistency of the images captured during a match.

Poor lighting conditions, for instance, can lead to underexposure or overexposure of the images, making it difficult to accurately identify the colors and features of team uniforms and the ball. This is crucial because the system relies on detecting these specific colors to classify objects within the image. Similarly, rapid changes in weather, such as sudden cloud cover or rain, can introduce noise and distortions in the images, such as glares and shadows, which may obscure crucial details needed for precise player and ball positioning.

Moreover, the varying angles and distances of cameras around the pitch can alter the perceived shapes and sizes of players and objects, complicating the task of consistently classifying and tracking them across different frames. This variation can lead to discrepancies in detecting the offside position, as the system might struggle to maintain a uniform standard for comparison and analysis across different game scenarios

Lastly, the system uses color quantization to simplify the image processing, but this method also has its drawbacks. Quantizing colors can lead to a loss of detail, making it difficult to distinguish between similar colors, such as different shades of a team’s uniform or the ball against a complex background. This can result in misclassification of players or the ball, leading to incorrect offside calls. More reliable systems might employ advanced machine learning algorithms that can learn to distinguish between subtle color differences and adapt to various lighting conditions. Beyond telling team jerseys apart on the field, colour differentiation also concerns the capacity to tell apart uniforms that are identical, especially when teams wear

colours that are quite similar. This challenge is compounded when players are distant or jersey details are blurred, potentially causing incorrect player identifications.

4.4 FRONT-END WITH REACT NATIVE

The creation of an interface for users to interact with was quite crucial. Although I managed to build the front end for the offside detection system using React Native, I was unable to fully integrate the mobile application with the image processing algorithm within the project timeline. The front end is designed to interact with the camera of a mobile device, allowing continuous capture of images during a football game. Here is a brief overview of the frontend functionalities:

1) **Camera Access and Permissions:** The app begins by requesting camera permissions from the user. Once granted, it allows the app to access the camera features of the device.

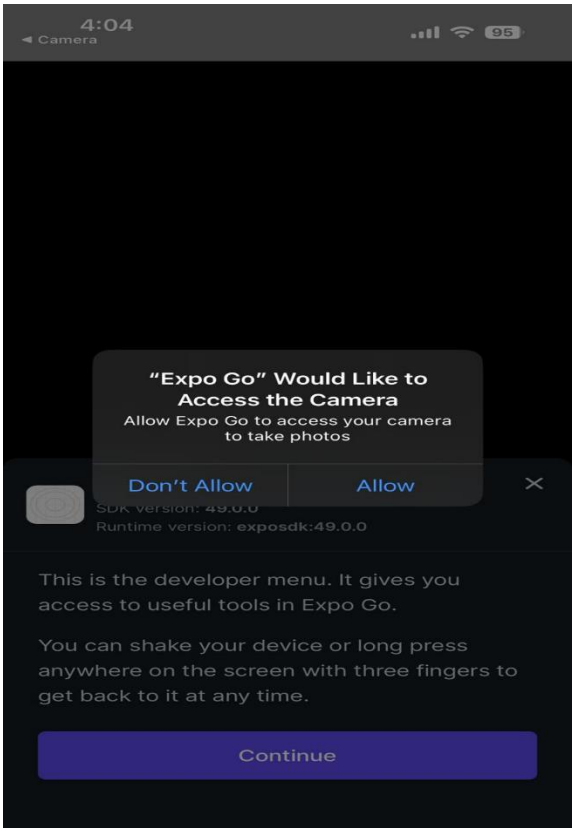


Figure 22: Mobile Camera Permission Request

2) **Image Capture Mechanism**: Users can start and stop the image capture process through a simple button interface. The app is set to automatically take pictures at intervals of three seconds when the capture function is active, mimicking a live game scenario where continuous monitoring is

required.

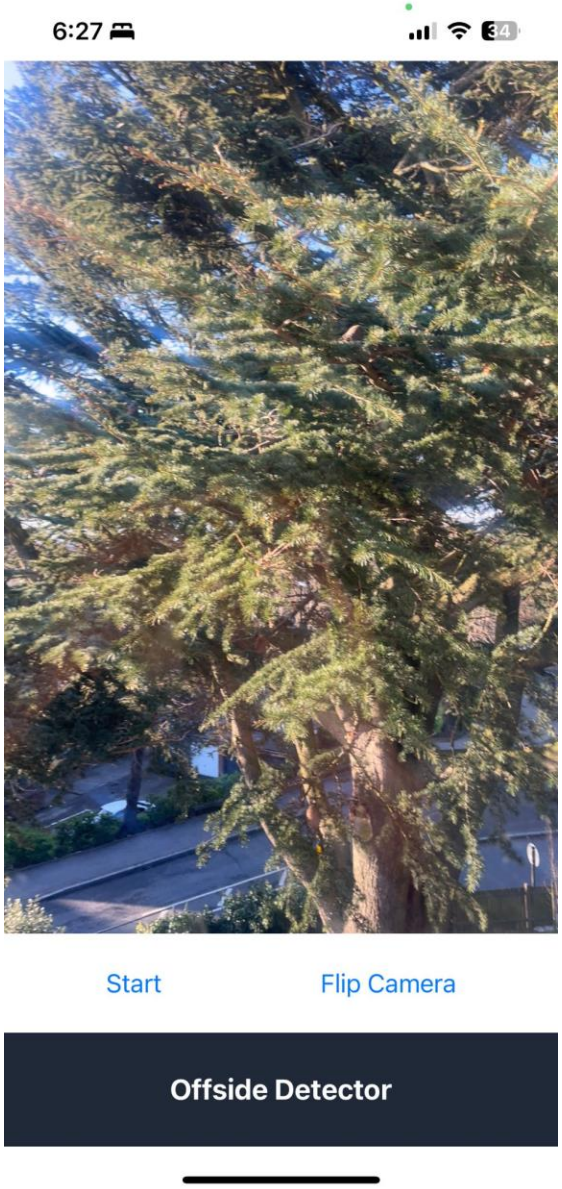


Figure 23: Home Screen

3) **Camera Switching:** The user can switch between the front and back cameras of the device to adapt to different use cases and preferences. This is accessed through a “Flip Camera” button.

4) **Image Processing Integration:** Each captured image is intended for processing to detect offside situations. However, this functionality is

currently represented by a placeholder function, `processImage`, and has not been fully implemented. The planned integration with the image processing algorithm remains an incomplete aspect of this project due to time constraints. This shortfall highlights a critical area for future development and integration efforts.

5) **User Feedback:** Upon detecting an offside condition, the app provides immediate feedback to the user through a Toast message, informing them of the detection and automatically stopping further image capture to address the event.

5. TESTING & EVALUATION

5.1 TESTING COLOURED SAMPLE PLAYERS

The system was first tested through a series of coloured sample players for ease of identification and to simplify the initial validation process. This testing approach allowed quick assessment of the system's ability to recognize and classify different entities on the pitch based on their assigned colors. The scenarios in this case are ignoring the goalkeeper and the goal post with the assumption they exist. Below are some of the test scenarios:

Scenario 1: Scenario 1 illustrates the transition where the first attacking player (red) passes the ball to the second attacking player (red) who is ahead of the last defender (yellow). This case is offside.

Images

Figure 24: Coloured Players Tests - Case 1

Figure 25: Coloured Players Tests - Case 2

Result

```
}
New player has possession in current frame
Offside detected in the transition from the first to the second image: true
Completed offside analysis.
```

Figure 26: Coloured Players Tests - Scenario 1 Result

Scenario 2: Scenario 2 illustrates the transition where the first attacking player (red) moves the ball ahead of the last defender (yellow) while the second attacking player (yellow) remains in the same position. This case is not offside.

Images

Figure 27: Coloured Players Tests - Case 1

Figure 28: Coloured Players Tests - Case 2

Result

```
    }
  }
  Same player retains the possession from the previous frame
  Offside detected in the transition from the first to the second image: false
  Completed offside analysis.

  C:\Users\User\Downloads\FootballDataAnnotationCode>
```

Figure 29: Coloured Players Tests - Scenario 2 Result

Scenario 3: Scenario 3 illustrates the transition where the first attacking player (red) passes the ball to the second attacking player (red) and moves ahead of the last defender (yellow). Hence, they are both ahead of the last defender. This case is offside.

Images

Figure 30: Coloured Players Tests - Case 1

Figure 31: Coloured Players Tests - Case 3

Result

```
}
New player has possession in current frame
Offside detected in the transition from the first to the second image: true
Completed offside analysis.
```

Figure 32: Coloured Players Tests - Scenario 3 Result

5.2 TESTING REALISTIC PLAYERS & EVALUATION

The offside detection system was assessed through a structured evaluation to ascertain its efficacy in real-match scenarios. The primary goal was to understand the system's performance across various offside and non-offside situations, using a range of photo-shopped football game simulations.

The system's accuracy was evaluated using a structured test set of 20 football game scenarios, where 5 were depicting offside and the other 5 represented legitimate onside positions.

The test results were compiled into a confusion matrix, which categorizes the system's predictions into four outcomes:

True Positive (TP): Offsides correctly identified as offside.
False Positive (FP): Onside situations incorrectly identified as offside.
True Negative (TN): Onside situations correctly identified as not offside.
False Negative (FN): Offsides that were missed by the system.

Table 1: Results for Sequence of Realistic Images

Scenario	Ruling	Outcome
Case A – B	Onside	TN
Case A – C	Offside	FN
Case A - D	Offside	FN
Case A - E	Onside	TN
Case A - F	Onside	TN
Case H - I	Offside	FN
Case H - G	Offside	FN
Case H - L	Onside	TN
Case J - K	Onside	TN
Case H - C	Offside	TP

Table 2: Realistic Images Count

Outcome	Count
True Positive	1
False Positive	0
True Negative	5
False Negative	4

5.4 SENSITIVITY AND SPECIFICITY ANALYSIS

The test results were compiled into a confusion matrix, which categorizes the system's predictions into four outcomes:

- **Sensitivity (True Positive Rate):** Measures the proportion of actual offsides correctly identified by the system.

- **Specificity (True Negative Rate):** Measures the system's ability to correctly identify non-offside scenarios.

Calculations for these metrics are based on the counts from the confusion matrix:

TP = 1

FN = 4

TN = 5

FP = 0

Sensitivity = TP / (TP + FN) = 1 / (1 + 4)

Specificity = TN / (TN + FP) = 5 / (5 + 0)

These calculations yield a sensitivity of approximately 20% and a specificity of 100%.

5.5 FUTURE WORK

The creation of a mobile application and the use of machine learning offer interesting opportunities for further improvement and wider applicability as the offside detection system advances. These developments aim to increase offside detection speed and accuracy while also providing users at all levels with access to advanced offside systems.

5.5.1 Improvement with Machine Learning

While considering all the challenges encountered in this project, the offside detection system's capabilities can be revolutionized through the integration of machine learning. Through model training on a variety of datasets that encompass a broad range of gaming scenarios, the system can acquire the ability to identify intricate patterns and complex variations in player locations and motions. This method would improve the system's ability to identify offsides, particularly in situations when there are varying player densities or partial occlusions.

5.5.2 Embedding into a Mobile Application

Although I successfully developed the front end of a mobile application designed to integrate this advanced offside detection system, time constraints prevented me from fully evolving it into a comprehensive mobile solution. The vision was to provide referees across low levels with a tool that could offer real-time insights and assist in making offside calls during football matches.

5.5.3 Comprehensive User Interface Design

A comprehensive user interface design would play a crucial role in the usability of the application. It should provide clear, concise, and immediate feedback regarding offside decisions. Features like instant replays, zoomed-in views, and graphical illustrations of player positions relative to the last defender and the ball can significantly enhance the user experience. Implementing interactive elements where users can view the offside line and player positions in real-time would also add educational value, helping users understand the decision-making process behind the scenes.

5.5.4 Enhancing Testing Protocols with Realistic Game Data

Future work should also focus on refining the testing protocols by integrating more realistic game data to improve the offside detection system. This involves collecting diverse game footage, including different player formations, and varied environmental conditions, to enhance the system's adaptability and accuracy in real-world scenarios. Utilizing high-definition, multi-angle camera setups will also enrich the dataset, providing a more comprehensive analysis of player positions and movements. Collaborating with football leagues to access extensive match recordings or organizing controlled simulated matches could be crucial in achieving these goals,

leading to a more effective tool for real-time offside decision-making.

5.5.5 Addressing Challenges

Several challenges must be addressed in future developments, including ensuring data privacy and security, especially when deploying cloud-based processing solutions. Additionally, the accuracy of player and ball detection in various environmental conditions, such as changing weather or lighting, must be reliable. Not only could this development lead to more accurate and fairer gameplay, but it also has the potential to transform the role of technology in sports, making sophisticated analysis tools more accessible to a broader range of sports professionals and enthusiasts. This progression requires continuous research, development, and collaboration between technologists, designers, and the sports community to ensure the solutions developed are effective, practical, and enhance the beautiful game of football.

6. CONCLUSION

The journey toward developing an effective offside detection system has been one of exploration, experimentation, and learning from failures. Initially, the ambition was to leverage state-of-the-art technologies like YOLO (You Only Look Once) for object detection and advanced pose estimation models to create a system that could automatically and accurately identify offside incidents in football matches. The appeal of these technologies lies in their proven capabilities for recognizing objects and human figures in complex visual scenes, promising a high degree of accuracy and real-time performance. However, the application of YOLO for detecting players and the ball presented significant challenges. The failure of YOLO was primarily due to its inability to differentiate between players from different teams using the pre-trained model, and the complexity and time constraints involved in retraining the model with team-specific data.

Faced with these challenges, it became evident that a different approach was needed. The decision to pivot to a rule-based analytical framework was driven by the necessity to find a more reliable and straightforward method that could overcome the limitations encountered with YOLO. This new strategy, focusing on color quantization and spatial analysis within a grid-based system, aimed to simplify the complex visual data of football matches into more manageable components. This shift was not just about adopting new tools but also about rethinking the problem of offside detection from different perspectives. By moving away from the heavy reliance on AI and machine learning models, a more pragmatic approach that prioritized simplicity, and reliability was embraced. In retrospect, the journey through the initial challenges with YOLO to the development of a rule-based system underscores the iterative nature of technological innovation. It highlights the importance of adaptability, the willingness to confront and learn from failures, and the continuous search for solutions that balance sophistication with

practical efficacy.

The image processing approach shows promise in automating offside detection in football matches, offering a significant step forward from traditional machine learning methods. However, to achieve the levels of accuracy required for practical application in professional settings, the system needs further refinement and testing. Future developments could explore hybrid approaches that combine rule-based methods with machine learning techniques to enhance decision accuracy, particularly in handling complex and ambiguous offside scenarios. As an undergraduate student who embarked on this journey, it offers a unique opportunity to contribute to an innovative field, bridging academic learning with real-world application and potentially changing how we experience and understand sports.

REFERENCES

1. Abdullah Hamdi, Jan Held, Anthoy Cioppa, Bernard Ghanem, S. G. Marc Van Droogenbroeck (2023). Video Assistant Referee System for Automated Soccer Decision Making from Multiple Views (VARS). University of Li`ege. [online] Available at: https://openaccess.thecvf.com/content/CVPR2023W/CVSports/papers/Held_VARS_Video_Assistant_Referee_System_for_Automated_Soccer_Decision_Making_CVPRW_2023_paper.pdf. [Accessed 21 Dec. 2023].
2. Iman Abaspur Kazerouni, Luke Fitzgerald, Gerald Dooly, Daniel Toal (June 2022). A Survey of State-Of-The-Art on Visual SLAM. Department of Electronics and Computer Engineering, Centre for Robotics and Intelligent Systems, University of Limerick. [online]. DOI: [https://www.sciencedirect.com/science/article/pii/S0957417422010156]. [Accessed 21 Dec. 2023].
3. Lu, F., Milios, E (1997). Globally Consistent Range Scan Alignment for Environment Mapping (Article). Department of Computer Science, York University. [online]. Available at: https://www.scopus.com/record/display.uri?eid=2-s2.0-0031249780&origin=inward&txGid=a11244444cbf6d6e72d48e21bc9cde04. [Accessed 21 Dec. 2023].
4. Saikot Chaterjee (June 2023). Use of Modern Technology in the recently concluded Qatar World Cup Football. Associate Professor and Head & Director Sports Ex-Office, Department of Physical Education, University of Kalyani. Vol: 8, Issue: 1. [online]. DOI: [https://doi.org/10.58914/ijyesspe.2023-8.1.6]. [Accessed 29 Dec. 2023].
5. Prof. Dr. Işık Bayraktar (Oct. 2023). The Use of Developing Technology in Sports. Özgür Publications. [online]. DOI: [https://doi.org/10.58830/ozgur.pub315]. [Accessed 29 Dec. 2023].
6. Arik Badami, Mazen Kazi, Sajal Bansal, Krishna Samdani (June 2019). Review On Video Refereeing using Computer Vision in

Football. The Institute of Electrical and Electronics Engineers. 2018 Punecon. DOI: [10.1109/PUNECON.2018.8745418]. [Accessed 27 Dec. 2023].

7. T. D'Orazio, Marco Leo, P. Spagnalo, P. L. Mazzeo, N. Mosca. M. Nitti, A. Disante (Dec. 2009). An Investigation into the Feasibility of Real-Time Soccer Offside Detection from a Multiple Camera System. IEEE Transactions on Circuits and Systems for Video Technology, vol. 19, no. 12, pp. 1804-1818. DOI: [10.1109/TCSVT.2009.2026817]. [Accessed 28 Dec. 2023].
8. Jabulani K. Makhubela, Tranos Zuva, Olusanya Yinka Agunbiade, Faculty of Applied and Computer Sciences, Vaal University of Technology, Vanderbijlpark, South Africa. A Review on Vision Simultaneous Localization and Mapping (VSLAM). The Institute of Electrical and Electronics Engineers. 2018 IEEE International Conference on Intelligent and Innovative Computing Applications ICONIC. [online]. Available at: https://ieeexplore.ieee.org/abstract/document/8601227. [Accessed 28 Dec. 2023].
9. Zhong-Qui Zao, Peng Zheng, Shou-Tao Xu, Xingdong Wu (2019). Object Detection with Deep Learning: A Review. IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, VOL.30, NO.11. [online]. Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8627998. [Accessed 2 Jan. 2024].
10. Zhengxia Zao, Keyan Chen, Zhenwei Shi, Yuhong Guo, Jieping Ye (2023). Object Detection in 20 Years: A Survey. Vol. 111, No. 3. PROCEEDINGS OF THE IEEE. [online]. Available at: [https://arxiv.org/abs/1905.05055v3]. [Accessed 8 Jan. 2024].
11. Souhail Gennouni, Ali Ahaitouf, Anass Mansouri (2014). Multiple object detection using OpenCV on an embedded platform. 2014 Third IEEE International Colloquium in Information Science and

Technology (CIST). [online] Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7016649. [Accessed 9 Jan. 2024].

12. H. Deshpande, A. Singh, H. Herunde (2020). Comparative Analysis on YOLO Object Detection with OpenCV. Department of Computer Application, Jain University, Bengaluru, Karnataka, India. Int. J. Res. Ind. Eng. Vol. 9, No. 1 (2020) 46–64. [online] Available at: https://www.riejournal.com/article_106905_afd0caf26202eb3ac3b605fd17894255.pdf. [Accessed 9 Jan. 2024].
13. *Jason Brownlee* PhD (2021). A Gentle Introduction to Object Recognition with Deep Learning. Article [online]. Available at: https://machinelearningmastery.com/object-recognition-with-deep-learning. [Accessed 10 Jan. 2024].
14. *Abhigith Abraham* (2023). Real-Time Object Detection Using YOLOv8: Step-by-Step Walkthrough. Article [online]. Available at: https://www.e2enetworks.com/blog/real-time-object-detection-using-yolov8-step-by-step-walkthrough. [Accessed 10 Jan. 2024].
15. React Native (2023). React Native Expo Documentation [online]. Available at: https://docs.expo.dev. [Accessed 9 Jan. 2024].
16. Roboflow (2023). Roboflow Documentation [online]. Available at: https://docs.roboflow.com. [Accessed 9 Jan. 2024].
17. OpenCV (2023). OpenCV Documentation: Python Implementation [online]. Available at: https://docs.opencv.org/master/d6/d00/tutorial_py_root.html. [Accessed 9 Jan. 2024].
18. YOLO (2023). YOLO (You Only Look Once) Documentation [online]. Available at: https://pjreddie.com/darknet/yolo. [Accessed 9 Jan. 2024].
19. Alex Kendall, Matthew Grimes, Roberto Cipolla (2015). PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalisation. [online] University of Cambridge. Available at:

https://openaccess.thecvf.com/content_iccv_2015/papers/Kendall_PoseNet_A_Convolutional_ICCV_2015_paper.pdf. [Accessed 3 Mar. 2024].

20. Yu Chen, Chunhua Shen, Xiu-Shen Wei, Lingqiao Liu, Jian Yang (2017). Adversarial PoseNet: A Structure-aware Convolutional Network for Human Pose Estimation. [online] Nanjing University of Science and Technology, University of Adelaide, Nanjing University. Available at: https://openaccess.thecvf.com/content_ICCV_2017/papers/Chen_Adversarial_PoseNet_A_ICCV_2017_paper.pdf. [Accessed 3 Mar. 2024].
21. Seyedamirreza Hesamian (2020). Deep Learning Models for 2D Tracking and 3D Pose Estimation of Football Players. Master's Thesis, Politecnico di Torino. Supervisor: Prof. Andrea Giuseppe Bottino. [online] Available at: https://webthesis.biblio.polito.it/15254/1/tesi.pdf. [Accessed 3 Mar. 2024].

APPENDICES

Figure 33: Realistic Images of Players - Case J

Figure 34: Realistic Images of Players - Case I

Figure 35: Realistic Images of Players - Case H

Figure 36: Realistic Images of Players - Case G

Figure 37: Realistic Images of Players - Case F

Figure 38: Realistic Images of Players - Case E

Figure 39: Realistic Images of Players - Case D

Figure 40: Realistic Images of Players - Case C

Figure 41: Realistic Images of Players - Case B

Figure 42: Realistic Images of Players - Case A

Figure 43: Realistic Images of Players - Case K

Figure 44: Realistic Images of Players - Case L