AUTOMATED BADMINTON SCORING SYSTEM USING COMPUTER VISION

MINI PROJECT REPORT

Submitted in partial fulfilment of the
Requirements for the award of Bachelor of Technology Degree
In Electronics and Communication Engineering
Of APJ Abdul Kalam Technological University

By

Vaishnav D Prabhath	**(MBT22EC105/B22EC1256)**
Ryan M Jacob	**(MBT22EC086/B22EC1248)**
Abin George	**(MBT22EC007/B22EC1204)**
Diya Vijayan	**(MBT22EC044/B22EC1225)**
Jaekob C John	**(MBT22EC057/B22EC1230)**

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING
MAR BASELIOS COLLEGE OF ENGINEERING & TECHNOLOGY
(Autonomous)
MAR IVANIOS VIDYANAGAR, NALANCHIRA, THIRUVANANTHAPURAM,695015.

2025

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

MAR BASELIOS COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous)

MAR IVANIOS VIDYANAGAR, NALANCHIRA, THIRUVANANTHAPURAM, 695015

MAR BASELIOS
COLLEGE OF ENGINEERING AND TECHNOLOGY
AUTONOMOUS

CERTIFICATE

This is to certify that this mini project report entitled **"AUTOMATED BADMINTON SCORING SYSTEM USING COMPUTER VISION"** is a bonafide record of work done by**Vaishnav D Prabhath, Ryan M Jacob, Abin George, Diya Vijayan and Jaekob C John** of the sixth semester Electronics and Communication branch towards the partial fulfillment of the requirements for the award of Bachelor of Technology Degree in Electronics and Communication Engineering of APJ Abdul Kalam Technological University.

Guide	Coordinator	Head of the Department
Dr. Vineetha Mathai	**Mr. Anoop K Johnson**	**Dr. Luxy Mathews**
Assistant Professor	**Assistant Professor**	**Associate Professor**
Dept. of ECE	**Dept. of ECE**	**Dept. of ECE**
MBCET	**MBCET**	**MBCET**

ACKNOWLEDGEMENT

With great enthusiasm and pleasure, we are bringing this mini project report to you. We use this opportunity to express our heartiest gratitude for the support and guidance offered to us from various sources during the completion of our project.

We are also grateful to **Dr. Luxy Mathews, Associate Professor and Head**, Department of Electronics and Communication Engineering, for her valuable suggestions. It is our pleasant duty to acknowledge our Mini project Coordinator, **Mr. Anoop K Johnson, Assistant Professor** in the Department of Electronics and Communication Engineering, who has guided and given supervision for the project work. Let us express our heartfelt gratitude to our guide, **Dr. Vineetha Mathai, Assistant Professor** in ECE dept. for helping us throughout the project.

Above all, we owe our gratitude to the **Almighty** for showering abundant blessings upon us. We also express our wholehearted gratitude to all our classmates who have expressed their views and suggestions about our projects and have helped us during the course of the project.

We extend our sincere thanks and gratitude once again to all those who helped us make this undertaking a success.

ABSTRACT

Traditional badminton scoring systems rely on human referees and manual scoreboard updates, which are prone to errors, inconsistencies, and subjectivity in fast-paced matches. While electronic line-calling systems exist in professional tournaments, they are expensive and not accessible for lower-level competitions, training centres, or recreational use. The absence of a cost-effective, automated system makes it difficult to ensure fairness and accuracy in scoring, leading to disputes and inefficiencies in game management.

This project proposes an automated badminton scoring system using computer vision to address these challenges. The system utilises real-time video processing, which includes court line detection and AI-based shuttlecock tracking to determine point allocation accurately. By integrating deep learning techniques for object recognition and motion tracking, it minimises human errors while enhancing decision-making speed. The methodology involves camera-based court monitoring, shuttlecock landing detection, and automated score updates, providing a low-cost, accessible alternative for accurate badminton scoring.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

The realm of sports has seen significant technological evolution over the years. From goal-line technology in football to Hawk-Eye systems in tennis and cricket, automation has brought precision and fairness to competitive environments. These advancements have not only helped in enhancing accuracy but also in reducing human error and bias.

Badminton, despite its global popularity, still relies heavily on manual umpiring and scorekeeping, especially in semi-professional settings. Due to the game's fast pace and the shuttlecock's speed, it is challenging for referees to make accurate judgments in real time. To combat this, several advanced systems have been developed, including commercially available technologies like Hawk-Eye and Instant Review systems, which are used in professional tournaments, offering precise shuttle tracking and in/out decisions. These systems use high-speed, multi-angle cameras and sophisticated algorithms to deliver accurate results. However, they are often prohibitively expensive and require complex installations, making them impractical for use in amateur or college-level settings. On the other hand, some open-source projects attempt to bring automation into recreational play, but they lack real-time capabilities, require manual inputs and struggle with accuracy due to inconsistent lighting and camera quality. These limitations highlight the need for a lightweight, cost-effective alternative that retains essential functionality while also minimising setup complexity.

Recent advancements in computer vision and deep learning have paved the way for innovative tracking solutions in sports applications. Notably, *TrackNet*, proposed by Z. Chen, X. Li and M. Xie, introduced a deep learning framework capable of tracking high-speed and tiny objects such as shuttlecocks in badminton. The model combines spatial and temporal features to accurately predict object positions across frames, offering a robust foundation for shuttle tracking tasks. Similarly, object detection algorithms like *YOLO* (You Only Look Once) have gained popularity due to their real-time performance and adaptability. These works serve as references in the development of accessible and responsive sports tracking systems, especially for non-professional scenarios where cost and simplicity are essential.

1.1 Key Features and Scope of the Project

This project is an attempt to develop a **proof-of-concept model for automating umpiring in badminton**. The system combines shuttlecock tracking with a simplified scoring mechanism to demonstrate the feasibility of real-time, automated decision-making.

1. **Shuttlecock Tracking Using AI**: At the core of the system is an AI-based shuttlecock tracking model that monitors shuttle movement during gameplay. The model is designed to detect significant events such as ground hits and trajectory changes.

2. **Simplified Scoring Algorithm**: A basic scoring algorithm has been devised to process the shuttle's position and determine the result of a rally. It evaluates whether the shuttlecock lands inside or outside the court boundaries and updates the score accordingly. To ensure ease of implementation, the logic has been scaled down compared to official badminton rules, while still maintaining the essence of the game.

3. **Minimal Hardware Setup**: The system is built to operate using a single camera and a low-power computing unit. This approach minimises cost and complexity, making the setup accessible and practical for smaller-scale use cases.

1. **Real-Time Output**: The scores are updated in real time and transmitted to an Arduino-powered LCD module, simulating an actual scoreboard. This provides immediate feedback and enhances the viewer and player experience.

2. **Cost-Effective and Modular**: Designed with affordability and modularity in mind, the system can serve as a baseline for further enhancements, including multi-camera support, improved accuracy, and integration of professional rule sets.

CHAPTER 2

TECHNOLOGY IN GENERAL

2.1 Block Diagram

To understand the technologies used in this project, it is essential to first look at the overall system architecture as shown in Fig 2.1. The project is structured around a modular approach, with each module handling a specific aspect of shuttlecock tracking and scoring in a badminton match.

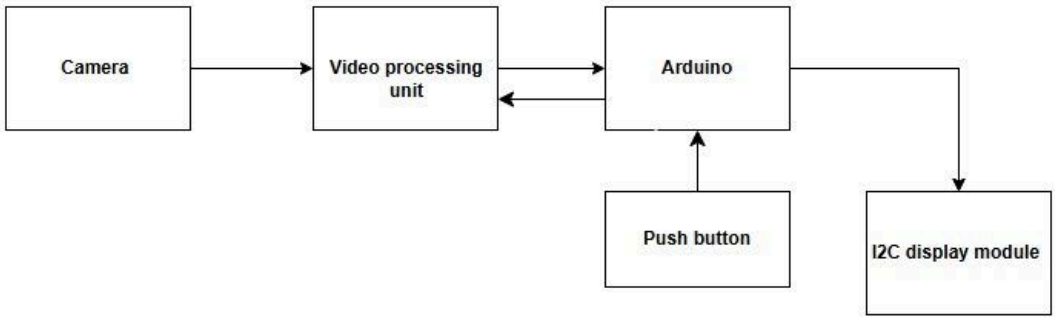

Fig 2.1 Block diagram representing the flow of data

2.2 Block Diagram Explanation

2.2.1 Camera Module

The camera module captures live footage of the badminton game. For optimal shuttlecock tracking and event detection accuracy, the camera should support at least 1080p resolution and a frame rate of 30–60 fps. The live video feed is passed in real-time to the Video Processing Module.

2.2.2 Video Processing Module

This is the system's core computational unit, hosted on a laptop. It comprises multiple submodules that process the incoming frames, as shown in Fig 2.2.

Fig 2.2 Submodules of Video Processing Module

a. Shuttlecock Tracking / Object Detection Module

Two object detection models were explored.

- **YOLOv8:** A real-time object detection model that processes each frame independently. YOLOv8 divides the image into a grid, predicting bounding boxes and class probabilities per cell. It is fast and efficient for live video but may struggle with brief occlusions or motion blur.

- **TrackNet V3:** A temporal tracking model that uses consecutive frames to predict the shuttlecock's location. It excels in handling motion blur, small object size, and temporary occlusions. However, it's more suited for offline or slightly delayed processing due to its temporal nature.

Table 2.1 Comparison Yolov8 vs Tracknet V3

Feature	YOLOv8	TrackNet V3
Detection Type	Real-time per-frame detection	Temporal tracking using frame history
Strength	Fast and lightweight	Robust tracking in occlusion/blurs
Limitation	Less accurate with fast motion	Not ideal for real-time responsiveness

b. Impact Detection Module

Monitors the shuttlecock's motion and detects when it abruptly stops or hits the ground. The logic is based on analysing sudden positional discontinuities in the tracked shuttlecock trajectory.

c. In/Out Detection Module

Verifies whether the shuttlecock's ground hit occurred in or out of the court boundaries. A predefined set of court coordinates (based on the camera's angle) is used to determine the location of the hit and identify which side of the court it occurred on, attributing the point to the correct player.

d. Score Decision Module

Based on the result from the In/Out module, this module determines if the score should be incremented. It then communicates with the Arduino Uno module to trigger the appropriate score update. This module only activates when a signal from the push button module is received, indicating an active rally is being played.

2.2.3 Arduino Uno Module

The Arduino Uno acts as the control unit responsible for managing the score logic. It receives score increment signals from the Video Processing Module and updates the internal score registers accordingly. Additionally, it monitors the Push Button Module and forwards its status to the video processor, controlling whether detection should be active.

2.2.4 Push Button Module

A simple interface connected to the Arduino Uno, the push button determines whether a rally is in progress. When pressed, the system is permitted to detect events and update scores. This manual control helps avoid false detections during breaks or unintentional movement on the court.

2.2.5 I2C LCD Display

The final score is displayed in real-time on a character LCD module interfaced with the Arduino via I2C protocol. I2C allows efficient two-wire communication (SDA and SCL), reducing pin usage while enabling fast and reliable updates to the display.

CHAPTER3
TECHNOLOGY IN SPECIFIC

3.1 Object Detection Models

3.1.1 YOLOv8 Shuttlecock Detection Model

YOLOv8 is one of the most advanced object detection architectures today, offering an impressive combination of speed and accuracy. It's especially useful for applications requiring real-time processing, such as live sports analytics, surveillance systems, and autonomous navigation. In our project, we adopted the YOLOv8s variant, which is optimised for high speed and lighter computational load, making it suitable for real-time shuttlecock detection in badminton matches.

The first step in building the shuttlecock detection system was dataset creation. We curated a dataset of over 6,000 images where the shuttlecock was visible in various contexts and lighting conditions. These images were labelled using Roboflow, where bounding boxes were drawn around the shuttlecock and assigned the appropriate class name. The labelled dataset was then augmented with transformations such as rotation, flipping, and brightness adjustment to enhance generalisation. This final dataset was split into training, validation, and test sets in a typical 70:15:15 ratio.

The YOLOv8 model was trained using PyTorch with a batch size of 8. The model was trained for around 50 epochs using the default learning rate settings. Throughout training, key metrics like precision, recall, and mAP (mean Average Precision) were monitored to ensure the model was learning effectively. Once trained, a confidence threshold of around 0.4 to 0.5 was applied during inference to filter out weaker detections.

3.1.2 TrackNetV3 Overview

TrackNetV3 is another deep learning model tailored for tracking small, fast-moving objects such as shuttlecocks or tennis balls. Unlike YOLO, which detects objects in single frames, TrackNetV3 focuses on the object's trajectory across multiple frames. It takes a sequence of three or more consecutive frames, extracts spatial and temporal features using a deep backbone like VGG16, and outputs a heatmap indicating the most probable location of the object as shown in Fig 3.1.

This model excels in scenarios where the object is extremely fast, small, or blurred—common issues in sports footage. However, TrackNetV3 has several drawbacks. Its need for multiple frame inputs and heavy architecture makes real-time integration challenging. The model also consumes significant GPU memory and struggles with missing frames, often caused by inconsistent frame capture.

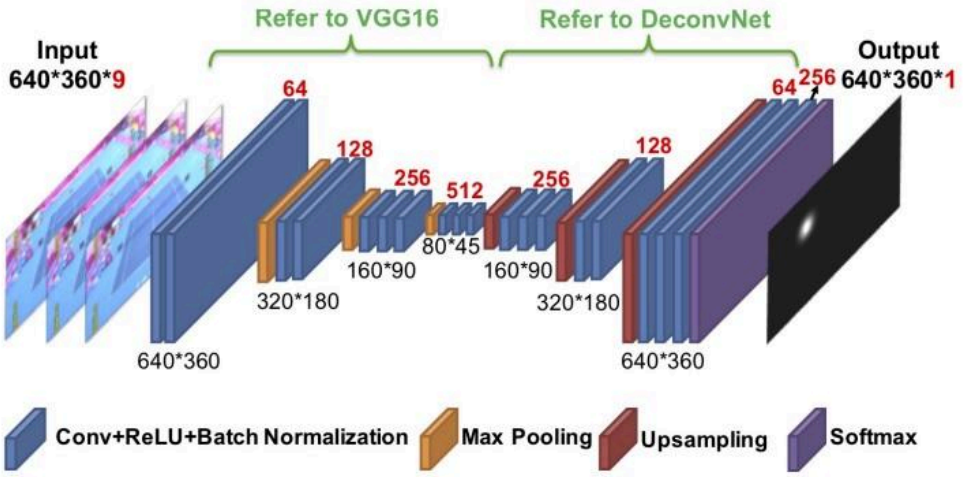

Fig 3.1 Architecture of Tracknet

3.1.3 Why YOLOv8s Was Chosen for Real-Time Video Processing

While TrackNetV3 offered high accuracy for object tracking, its real-time feasibility was limited due to its computational complexity and frame dependencies. YOLOv8s, on the other hand, operates on single frames and delivers much faster inference speeds. The lightweight nature of YOLOv8s, combined with its robustness and adaptability, made it the ideal choice for our project’s live shuttlecock detection module.

3.2 Court Marking Logic and Impact Detection

To determine whether the shuttlecock landed "in" or "out," we first required a precise mapping of the badminton court. This was done manually at the start of the game by selecting key points on the court using a visible frame with minimal player movement. The user marks the outer boundaries, the net line and the sides under the 2 teams. These selected points are used to construct reference court lines, assuming a fixed camera angle and minimal distortion.

Once the court is mapped, the system identifies impact points by analysing the shuttlecock's abrupt motion halt in consecutive frames. These suspected impact points are checked against the court boundaries to classify them as either "IN" or "OUT." This logic is entirely geometry-based and assumes a rectangular court layout, enabling fast and efficient decision-making during gameplay. This manual court setup ensures accuracy under various conditions, especially when court markings are faded or lighting is uneven.

3.3 Arduino Uno Functionalities and Serial Communication

The Arduino Uno serves as the bridge between our detection system and the physical scoreboard. It communicates with the laptop through serial communication using a USB cable. When a rally is won, Python code running on the PC sends simple string commands such as "P1+" or "P2+" over the serial port. The Arduino listens continuously using the Serial.read() function inside its main loop and updates scores accordingly.

The Arduino also manages the game logic. Every time a player scores a point, it increments the score. Upon reaching the set-winning score, the set count is updated, and the points are reset for the next set. Once a player wins the majority of sets (e.g., two out of three), the game ends, and a winner is declared. This logic is hard-coded into the Arduino's firmware, ensuring consistency and independence from the host PC during a match.

Fig 3.2 Arduino Uno

3.4 LCD Display Integration

To visually represent scores and game status, we integrated two 16x2 LCD with the Arduino. Instead of using multiple digital pins for each LCD signal line, an I2C interface was used as shown in Fig 3.3. This reduces wiring complexity and requires only four connections: VCC, GND, SDA, and SCL. The LCD is updated in real-time every time a serial command is received and processed.

This setup ensures that players and spectators can easily follow the current score, sets won, and game status. The use of the I2C interface not only simplifies the hardware but also makes the display updates faster and more reliably.

Fig 3.3 LCD integrated using I2C module

CHAPTER4

HARDWARE IMPLEMENTATION

4.1. Circuit Diagram

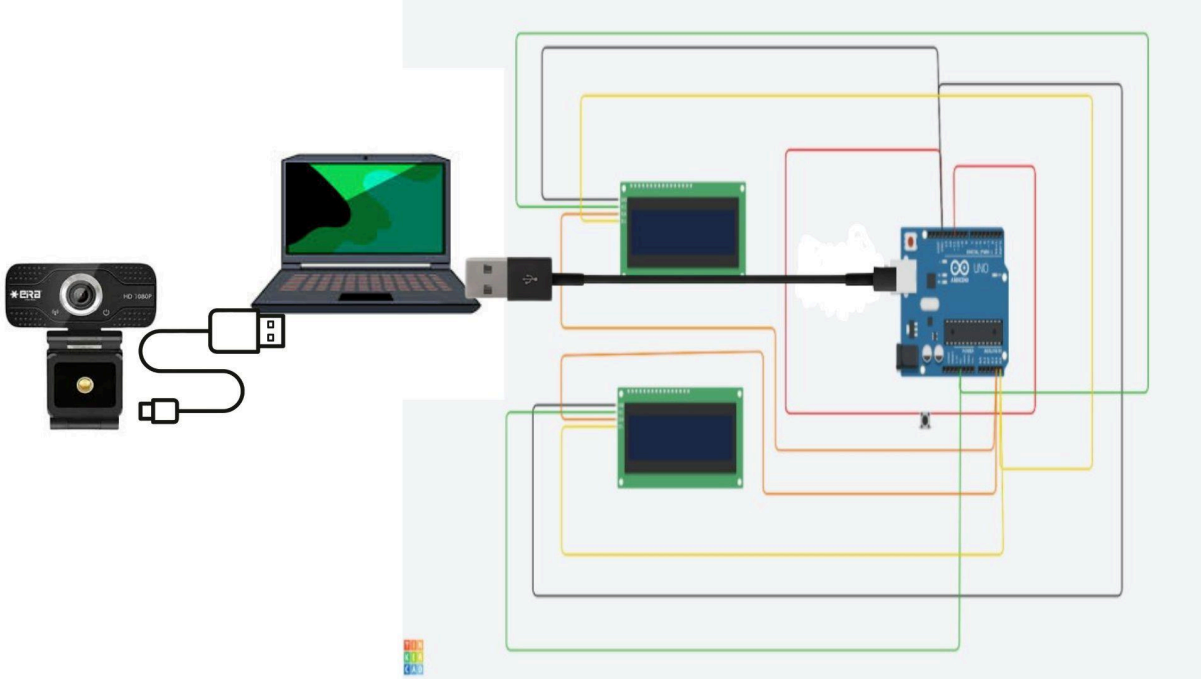

Fig 4.1 Circuit diagram

4.2. List of Components

4.2.1 Camera

- Interface: USB
- Connected to: Laptop
- Function: Captures real-time video feed for shuttlecock detection.

4.2.2 Laptop

- GPU (Nvidia RTX3050): Video processing and AI model detection
- Interface: Camera and Arduino connecting via USB
- Function: Platform for running Python and AI processing

4.2.3 Arduino Uno

- Interface: USB
- Connected to: Laptop
- Function: Acts as the scoreboard controller, receives serial data from the Python script to display player scores and game status on LCDs.

4.2.4 LCD Displays (2 x 16x2 with I2C)

- Type: 16x2 I2C LCD Module
- VCC: Connected to Arduino 5v
- GND: Connected to Arduino GND
- SDA: Connected to Arduino A4 (I2C SDA)
- SCL: Connected to Arduino A5 (I2C SCL)
- Function: Displays live scores of both players.

4.2.5 Push Button (4-Pin)

- Type: Tactile push button
- Connected Pins:
    - One diagonal pair shorted together to GND
    - The other diagonal pair connected to a digital input pin on the Arduino (e.g., pin 2) and pulled high using an internal pull-up resistor
- Function: Used for match reset or manual override.

4.3. Working

4.3.1 Real-Time Video Processing

- The video stream is captured through the USB webcam connected to the laptop.
- A Python script runs on the laptop, utilising a YOLO-based model to detect shuttlecock movements and identify ground hits.
- Based on the shuttlecock's position and motion, the system determines the scoring event.

4.3.2 Score Transmission

- Once a scoring event is detected, the Python script processes the event logic and updates the score.
- The updated score and game information is sent over the serial USB interface to the Arduino Uno.

4.3.3 Score Display

- The Arduino receives the updated score values and displays them on the two I2C LCDs.
- One LCD shows Player 1's and Player 2's scores, while the other shows the sets won by both.
- The display updates in real time as the game progresses.

4.3.4 Manual Control

- The push button connected to the Arduino provides a manual control feature.
- The ground hit detection is activated only when the push button is pressed.

4.4. Power Supply

- Both the Arduino Uno and the camera draw power from the laptop's USB ports.
- No external power supply is required.

CHAPTER5

SOFTWARE IMPLEMENTATION

5.1 Software Flow Diagram

Fig 5.1 shows the software flow of the prototype, starting from video acquisition to displaying the score. The detailed algorithm of the individual blocks is given in the next section.

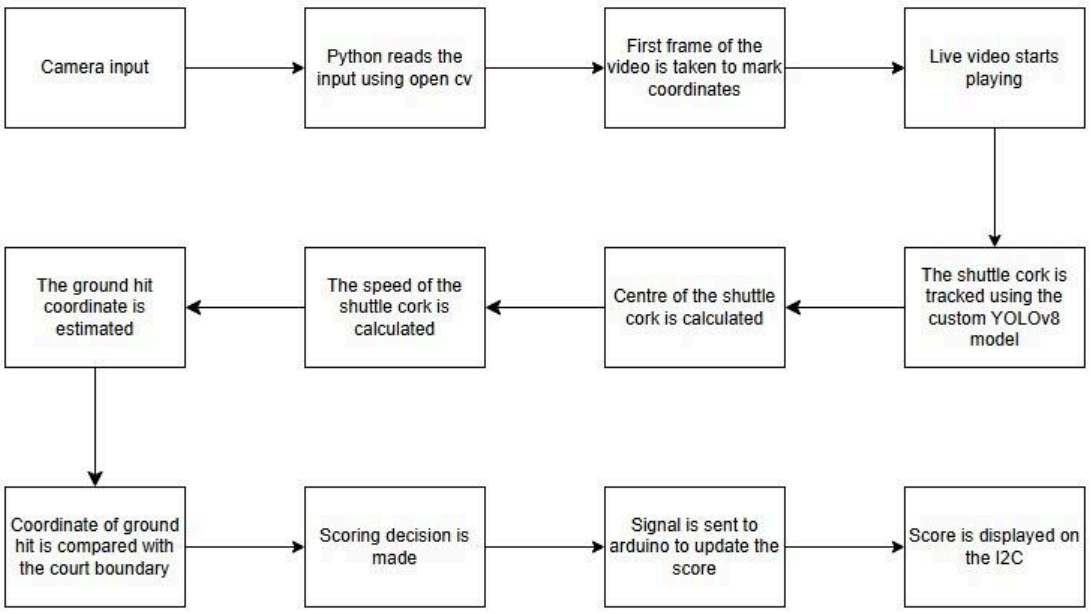

Fig 5.1 Software Flow Diagram

5.2 Algorithm

5.2.1 Court and Net Setup (User Input)

- Read the first frame of the video
- Use mouse clicks to collect the 6 court boundary points and 4 net points
- 6 court boundary points are stored in (polygon_points) as shown in Fig 5.2.
- Points 1-4 define player 1's court area
- Points 3-6 define player 2's court area
- 4 net points are stored in (net_points). These points define two halves of the court and the net region.

Fig 5.2 Court Boundary points, as stored in Polygon_points

5.2.2 Shuttle Detection

- Each frame from the input video is taken using Opencv
- Using the custom-made YOLO model presence of the shuttle cork is detected
- For each detected Shuttle cork
- Draw a bounding box and centre point
- Plot this bounding box in the output video display

5.2.3 Ground Hit Detection

- The centre of the shuttle cork in each frame is calculated
- Compute movement speed using the previous centre point and FPS.
- If speed is within a realistic bounce threshold (e.g., 0 < speed < 18 m/s) and the push button has been activated, it is a new ground hit point
- Set k = 1(preventing other ground hits to me marked until the start of the next serve)
- Store the x and y coordinates of the centre of the shuttle cork in the frame where the ground hit has been confirmed
- Plot this point on the output display

5.2.4 Scoring logic

- If the ground hit is in player1's court, a point is awarded to player2 and a serial data "1" is sent to Arduino
- If the ground hit is in player2's court, a point is awarded to player1 and a serial data "2" is sent to Arduino

- If the ground hit is outside both players' court but below the centre line, a point is awarded to player1 and a serial data “1” is sent to Arduino
- If the ground hit is outside both players' court but above the centre line, a point is awarded to player2 and a serial data “2” is sent to Arduino
- If the ground hit is in the polygon defining the net, then the string “0” is sent to Arduino

5.2.5 Python-Arduino Serial Communication

Sending Score to Arduino

- After detecting a valid ground hit and determining the score
- Send "1" if Player1 scores.
- Send "2" if Player2 scores.
- Send "0" if it's a net hit
- Always send the message with a newline character (\n) for Arduino to recognise the end.

Receiving Input from Arduino

- Continuously check if there is any incoming serial data.
- If the input is "BUTTON_PRESSED”
- Reset the bounce tracking logic (k = 0), indicating that the next serve is about to begin
- Prepare to detect the next valid ground hit.

5.2.6 Score processing logic in Arduino

- If the received input via serial communication is “1”, then the value of player1’s score is incremented by one
- If the received input via serial communication is “2,” then the value of player 2’s score is incremented by one
- Other wise there is no increment made, and Arduino waits for the next serial data
- It then checks for any update in the score from the previous score, and then updates the display

- When the score of either player1 or player2 reaches 11 the the number of sets played is increased by one for the respective player
- The scores of both players are reset to 0
- If the sets won by either player reaches 2, then a display message is generated showing “game over” along with the winning player's name as shown in Fig 5.3

Fig 5.3 Scoreboard

CHAPTER 6

RESULTS AND DISCUSSIONS

This chapter provides a comprehensive overview of the system's functionality, accuracy, and demonstration results. The outcomes highlight the key aspects of shuttlecock tracking, hit detection, and score computation using a single-camera setup integrated with an Arduino-based Scoring and display system. The key outcomes of the “Automatic Badminton Scoring System Using Computer Vision” are given below.

6.1 Testing Environment

- Indoor lighting with a fixed-angle single camera.
- Pre-fed court dimensions, calibrated to video frame.
- Court background with limited noise (Zero spectator movements and only one shuttlecock in the video feed at all times).
- Shuttlecourt background of a unique uniform colour.

6.2 Real-Time Detection and Ground Hit Identification

Shuttlecock was successfully detected in each video frame using YOLOv8 and TrackNetV3 object detection models as shown in Fig 6.1 and Fig 6.2, respectively. Ground hit detection is performed by analysing abrupt stoppage in the shuttle movement, with the assumption that the shuttle comes to rest only on ground hits.

Fig 6.1 Shuttlecock detected frame using Yolov8

Fig 6.2 Shuttlecock detected frame using Tracknet

6.3 Real-Time Scoring and Game Logic Execution

Score increment is triggered upon a valid ground hit detection. The scoring is done in reference to the manually added coordinate mapping of the badminton court, with accurate side-based scoring (Player 1 / Player 2 zone split based on camera view). The score increment data is then serially communicated to the Arduino.

Scoring logic has been simplified for ease of computation. A player who accumulates 12 points wins the set, and once a player wins two sets, they are declared the winner of the match.

6.4 System Performance and Evaluation

The developed automation prototype successfully demonstrates an integrated system that combines AI-based shuttlecock detection with embedded score computation and real-time display. It achieves real-time tracking and scoring, with detected scores and set/match logic accurately displayed through an Arduino-driven LCD module. The system operates at approximately 15 FPS on CPU and up to 30 FPS on GPU using YOLO, with instantaneous response from the Arduino and negligible display latency, ensuring a visually real-time experience. Detection accuracy for ground hits reaches around 70% in test conditions, although performance can be affected by factors such as blurred or occluded frames, camera angle, lighting, and court calibration. While the current system is not yet robust enough for match-level deployment, it serves as a functional testbed for scalable sports automation solutions. Despite some limitations, the project successfully meets its objective of delivering a working and demonstrable solution for mini-project evaluation.

CHAPTER 7
CONCLUSION

This project set out to explore the feasibility of using real-time technologies to automate the traditionally manual process of scoring in badminton. Through a single-camera setup and AI-based models, the system successfully demonstrated the potential of real-time shuttlecock tracking and automated scoring. The objective was to create a low-cost, accessible prototype capable of offering a more objective and efficient alternative to current methods, and this goal was effectively achieved.

The final system integrated the YOLO object detection model for shuttlecock tracking, chosen for its real-time processing capability. Although it faced accuracy drops during fast rallies and occlusions, it remained the most practical option under current hardware constraints. Ground hit detection was implemented using abrupt motion stop logic, and in/out decisions were evaluated through manually configured court boundaries. The Arduino Uno and I2C LCD module allowed for near-instantaneous score updates, completing a full pipeline from video input to scoreboard output.

Performance testing revealed a ground hit detection accuracy of approximately 70% under controlled conditions, with YOLO delivering frame rates of 15 FPS on CPU and 30 FPS on GPU. The Arduino module responded instantly, and the display latency was negligible, ensuring a visually real-time experience. While not ready for professional match deployment, the system demonstrates a reliable proof of concept and successfully fulfills the core goals of the mini-project.

This prototype lays a strong foundation for future enhancements. Multi-camera setups could resolve issues related to occlusion and side detection, while trajectory prediction and advanced court calibration could significantly boost accuracy and analytical depth. By combining embedded systems and AI in a unified framework, this project opens promising avenues for innovation in sports automation, particularly in making game officiation more transparent, consistent, and scalable.

REFERENCES

[1] Z. Chen, X. Li, and M. Xie, "TrackNet: A deep learning network for tracking high-speed and tiny objects in sports applications," *in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Long Beach, CA, USA, Jun. 2019, pp. 341-348, doi: 10.1109/CVPRW.2019.00058.

[2]A. S. P. A. R. Subramanian, "Object detection and tracking using YOLO," *in Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Montreal, QC, Canada, Oct. 2021, pp. 5987-5996, doi: 10.1109/ICCV48922.2021.00586.

[3]N. Dardagan, A. Brđanin, D. Džigal, and A. Akagic, "Multiple object trackers in OpenCV: A benchmark," *arXiv preprint arXiv:2110.05102*, Oct. 2021. [Online]. Available: https://arxiv.org/abs/2110.05102

[4]Badminton World Federation, "Simplified Rules of Badminton," Dec. 2015. [Online]. https://system.bwfbadminton.com/documents/folder_1_81/Regulations/Simplified-Rules/Simplified%20Rules%20of%20Badminton%20-%20Dec%202015.pd

[5]https://docs.arduino.cc/hardware/uno-rev3

[6]https://lastminuteengineers.com/i2c-lcd-arduino-tutorial/

APPENDIX

Program:
Python code for video footage tracking

```
import cv2
import serial
import time
import torch
import numpy as np
import math
import matplotlib.path as mpltPath
from ultralytics import YOLO
from shapely.geometry import Polygon, box

# Initialize video capture
cap = cv2.VideoCapture("inputvideo/mbcettest.mp4")

#ap.set(cv2.CAP_PROP_FRAME_HEIGHT, 1080)
#cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1920)
frame_size = (1400, 720
              )
frame_poly = box(0, 0, frame_size[0], frame_size[1])
img = np.ones((frame_size[1], frame_size[0], 3), dtype=np.uint8) * 255

# Configure the serial port
PORT = 'COM16'  # Replace with your Arduino's serial port
BAUD_RATE = 9600

# Serial port initialization
try:
```

```
    ser = serial.Serial(PORT, BAUD_RATE, timeout=1)  # Added timeout
    time.sleep(2)  # Give the serial port time to initialize
    print(f"Connected to serial port {PORT}")
except serial.SerialException as e:
    print(f"Error: Could not connect to serial port {PORT}: {e}")
    ser = None

def draw_polygon(img, polygon, color, thickness=2):
    if polygon.geom_type == "Polygon":
        contours = [np.array(polygon.exterior.coords, dtype=np.int32)]
    elif polygon.geom_type == "MultiPolygon":
        contours = [np.array(poly.exterior.coords, dtype=np.int32) for poly in polygon.geoms]
    else:
        return

    cv2.polylines(img, contours, isClosed=True, color=color, thickness=thickness)

def send_score(score):
    global ser
    if ser:
        try:
            ser.write(f"{score}\n".encode('utf-8'))
            print(f"Sent score: {score} to Arduino")
        except serial.SerialException as e:
            print(f"Error sending score: {e}")

fps = cap.get(cv2.CAP_PROP_FPS)

# Load the YOLO model with GPU support
```

```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
model = YOLO("ptfiles/best (1).pt").to(device)

prev_centr = None
speed = 0
points = []
polygon_points = []
net_points = []
p1 = 0
p2 = 0
score = "0000"
k = 1
frame_count = 0
c = False

def is_point_in_quadrilateral(point, quad):
    path = mpltPath.Path(quad)
    return path.contains_point(point)

def mouse_callback(event, x, y, flags, param):
    global polygon_points, first_frame
    if event == cv2.EVENT_LBUTTONDOWN and len(polygon_points) < 6:
        polygon_points.append((x, y))
        cv2.circle(first_frame, (x, y), 2, (0, 255, 0), 2)
        cv2.imshow("Select Points", first_frame)

def mouse_callback1(event, x, y, flags, param):
    global net_points, first_frame
    if event == cv2.EVENT_LBUTTONDOWN and len(net_points) < 4:
```

```
        net_points.append((x, y))
        cv2.circle(first_frame, (x, y), 2, (0, 255, 0), -1)
        cv2.imshow("Select Points", first_frame)

# Read the first frame and select points
i = 1
while i < 3:
    ret,  first_frame= cap.read()
    #first_frame = cv2.resize(first_frame, (1440, 720))
    if not ret:
        print("Failed to read the video")
        cap.release()
        break
    cv2.imshow("Select Points", first_frame)
    if i == 1:
        cv2.setMouseCallback("Select Points", mouse_callback)
        while len(polygon_points) < 6:
            cv2.waitKey(1)
    else:
        cv2.setMouseCallback("Select Points", mouse_callback1)
        while len(net_points) < 4:
            cv2.waitKey(1)
    i += 1

cv2.destroyWindow("Select Points")
cap = cv2.VideoCapture("inputvideo/mbcettest.mp4")
while cap.isOpened():
    ret, img = cap.read()
    #img = cv2.resize(img, (1440, 720), interpolation=cv2.INTER_CUBIC)
```

```
if not ret:
    break

# Read serial input (non-blocking)
if ser and ser.in_waiting > 0:
    line = ser.readline().decode('utf-8').strip()
    if line == "BUTTON_PRESSED":
        k = 0
        c = False
        print("Button Pressed! k set to 0.")

# Run the model on the current frame (GPU)
results = model(img)
net_pts = np.array([
       [303, 317],
       [635, 303],
       [639, 265],
       [313, 279]
      ])

for result in results:
    for box_data in result.boxes:
        x1, y1, x2, y2 = map(int, box_data.xyxy[0])
        confidence = box_data.conf[0].item()
        cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2)
        Cx, Cy = (x1 + x2) / 2, (y1 + y2) / 2
        cv2.circle(img, (int(Cx), int(Cy)), 5, (0, 0, 255), -1)
        confidence_text = f"{confidence:.2f}"
          cv2.putText(img, confidence_text, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX,
0.5, (255, 255, 255), 2)
```

```
        if prev_centr is not None:
            dx, dy = Cx - prev_centr[0], Cy - prev_centr[1]
            distance = math.sqrt(dx**2 + dy**2)
            speed = distance * fps
                                        cv2.putText(img, f"Speed: {speed:.2f} m/s", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)


            if 0 < speed < 18 and k == 0:
               k = 1
               c=True
               #if p1 ==11 or p2 == 11:
                 #p1=0
                 #p2=0
               NET = is_point_in_quadrilateral((Cx, Cy), net_pts)
               points.append((Cx, Cy))
               if is_point_in_quadrilateral((Cx, Cy), pts_tuple1):
                  #p2 += 1
                  score="2"
                  print("OUT1")
               elif is_point_in_quadrilateral((Cx, Cy), pts_tuple2):
                  #p1 += 1
                  score="1"
                  print("OUT2")
                   elif is_point_in_quadrilateral((Cx, Cy), new_polygon_tuple) and not NET and (Cy < pts[2, 1] or Cy < pts[3, 1]):
                  #p2 += 1
                  score="2"
                  print("OUT3")
                   elif is_point_in_quadrilateral((Cx, Cy), new_polygon_tuple) and not NET and (Cy > pts[2, 1] or Cy > pts[3, 1]):
```

```
                #p1 += 1
                score="1"
                print("OUT4")
            else:
                score="0"
                print("NET")
            send_score(score)
            #score=str(p1) + str(p2)
            #print(score)



        prev_centr = (Cx, Cy)
    if c== True:
        text_to_display = "OUT" if (is_point_in_quadrilateral((Cx, Cy), new_polygon_tuple)) else "In" if (is_point_in_quadrilateral((Cx, Cy), pts_tuple1)) else "IN" if (is_point_in_quadrilateral((Cx, Cy), pts_tuple2)) else "NET"
        cv2.putText(img, text_to_display, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)  # Place text at (10, 30)
    for point in points:
        cv2.circle(img, (int(point[0]), int(point[1])), 5, (255, 0, 0), -1)


    frame_count += 1
    #if frame_count % 30 == 0:
        #send_score(score)
        #if p1 == 11 or p2 == 11:
            #p1=0
            #p2=0


    if len(polygon_points) == 6 and len(net_points) == 4:
        pts1 = np.array(polygon_points, np.int32)
```

```
print(pts1)
net1= np.array(net_points, np.int32)
print(net1)
pts  = np.array([
  [104, 454],
  [818, 440],
  [646, 353],
  [299, 360],
  [349, 330],
  [585, 323]
 ])
first_half = pts[0:4, :]
second_half = pts[2:6, :]
pts_tuple1 = [tuple(pt) for pt in first_half]
pts_tuple2 = [tuple(pc) for pc in second_half]
new_polygon = frame_poly.difference(Polygon(pts_tuple1).union(Polygon(pts_tuple2)))
    new_polygon_tuple = tuple(map(tuple, np.array(new_polygon.exterior.coords))) if new_polygon.geom_type == "Polygon" else [tuple(map(tuple, np.array(poly.exterior.coords))) for poly in new_polygon.geoms] if new_polygon.geom_type == "MultiPolygon" else None
   cv2.polylines(img, [first_half.reshape((-1, 1, 2))], isClosed=True, color=(0, 0, 255), thickness=1)
  cv2.polylines(img, [second_half.reshape((-1, 1, 2))], isClosed=True, color=(255, 0,0), thickness=1)
    #cv2.polylines(img, [net_pts.reshape((-1, 1, 2))], isClosed=True, color=(0, 255, 0), thickness=1)
if new_polygon_tuple:
   draw_polygon(img, new_polygon, (255, 0, 0), 2)
cv2.imshow("Frame", img)
if cv2.waitKey(1) & 0xFF == ord("q"):
```

```
        break


cv2.destroyAllWindows()

cap.release()

if ser:

 ser.close()
```

Arduino Code:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

#define BUTTON_PIN 11

LiquidCrystal_I2C lcdSets(0x27, 16, 2);   // Set tracker LCD
LiquidCrystal_I2C lcdScore(0x26, 16, 2);  // Score display LCD

int scoreA = 0, scoreB = 0;
int setsWonA = 0, setsWonB = 0;
int currentSet = 1;
bool gameOver = false;
int setsPlayed = 0;
int newScoreA=0;
int newScoreB=0;
String cmp1="1";
String cmp2="2";
String scoreString="";

void setup() {
  lcdSets.init();
  delay(500);
  lcdSets.backlight();
  lcdSets.setCursor(0, 0);
  lcdSets.print("Current Set: 0");
  lcdSets.setCursor(0, 1);
  lcdSets.print("A :");
  lcdSets.print("B :");

  lcdScore.init();
  lcdScore.backlight();
  lcdScore.setCursor(0, 0);
  lcdScore.print("Player 1: 0");
  lcdScore.setCursor(0, 1);
  lcdScore.print("Player 2: 0");
```

```
  pinMode(BUTTON_PIN, INPUT_PULLUP); // Internal pull-up
  Serial.begin(9600);
}

void updateSetLCD() {
  lcdSets.setCursor(13, 0);
  lcdSets.print(setsPlayed); // Current set number
  lcdSets.setCursor(0, 1);
  lcdSets.print("A :");
  lcdSets.print(setsWonA);
  lcdSets.print("B :");
  lcdSets.print(setsWonB);
}

void updateScoreLCD() {
  lcdScore.setCursor(0, 0);
  lcdScore.print("Player 1: ");
  lcdScore.print(scoreA < 10 ? "0" + String(scoreA) : String(scoreA));
  lcdScore.setCursor(0, 1);
  lcdScore.print("  Player 2: ");
  lcdScore.print(scoreB < 10 ? "0" + String(scoreB) : String(scoreB));
}

void resetSet() {
  scoreA = 0;
  scoreB = 0;
  currentSet++;
  updateScoreLCD();
  delay(1000);
}

void button_press() {
  Serial.println("BUTTON_PRESSED");
}

void loop() {
  static unsigned long lastButtonPress = 0;
  static bool buttonPressed = false;
  unsigned long currentMillis = millis();

  // Handle incoming score
  if (Serial.available()) {
    String scoreString = Serial.readStringUntil('\n');
     // Clean any whitespace or \r


     //String part1 = scoreString.substring(0, 2);
     //String part2 = scoreString.substring(2, 4);
     //newScoreB = part2.toInt();
     //newScoreB = part2.toInt();
```

```
    if(scoreString==cmp1)
    {
     newScoreA=newScoreA+1;
    }
    if(scoreString==cmp2)
    {
     newScoreB=newScoreB+1;
    }
    else{
     newScoreA=newScoreA+0;
     newScoreB=newScoreB+0;
    }
    if (newScoreA != scoreA || newScoreB != scoreB) {
     scoreA = newScoreA;
     scoreB = newScoreB;
     updateScoreLCD();
    }

}

// Handle button
if (digitalRead(BUTTON_PIN) == LOW) {

  if (!buttonPressed && currentMillis - lastButtonPress > 300) {
    button_press();
    lastButtonPress = currentMillis;
    buttonPressed = true;
  }
} else {
  buttonPressed = false;
}

// Set win logic
if (!gameOver) {
  if (scoreA == 11) {
    setsWonA++;
    setsPlayed++;
    scoreA=0;
    scoreB=0;
    newScoreA=0;
    newScoreB=0;
    updateSetLCD();
    resetSet();
  } else if (scoreB == 11) {
    setsWonB++;
    setsPlayed++;
    scoreA=0;
    scoreB=0;
    newScoreA=0;
    newScoreB=0;
```

```
      updateSetLCD();
      resetSet();
    }

    if (setsWonA == 2 || setsWonB == 2) {
      gameOver = true;
      lcdSets.clear();
      lcdSets.setCursor(0, 0);
      lcdSets.print("Game Over");
      lcdSets.setCursor(0, 1);
      lcdSets.print(setsWonA == 2 ? "Winner: Team A" : "Winner: Team B");
      lcdScore.clear();
      lcdScore.setCursor(0, 0);
      lcdScore.print("Final Score");
      lcdScore.setCursor(0, 1);
      lcdScore.print("A:" + String(scoreA) + " B:" + String(scoreB));
      delay(5000);
}
}


}
```