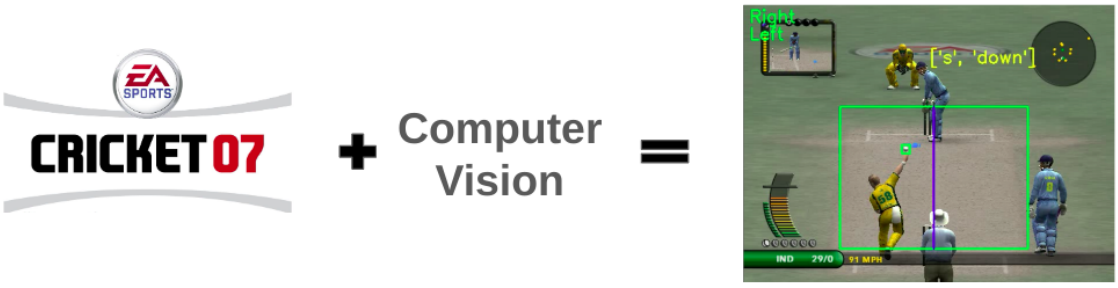
EA Sports Cricket 07: Transforming Gameplay Through Vision Automation

Our Presentation Video Link



Uday Kanth Reddy Kakarla
Department of Computer Science
University of Illinois Urbana-Champaign
uk3@illinois.edu

Girija Manoj Kumar Reddy Kalakoti
Department of Computer Science
University of Illinois Urbana-Champaign
gmk6@illinois.edu

Santosh Kumar Chejarla
Department of Computer Science
University of Illinois Urbana-Champaign
santosh8@illinois.edu

ABSTRACT

This report explains our approach to automating the gameplay in EA Sports Cricket 07 through the use of computer vision, machine learning, and heuristic methods. The project's main focus is on automating key decision-making processes thereby automating the batting, which includes the ball detection and making strategic batting decisions.

The methodology employs template matching learned in the class for precise ball detection, and a combination of neural networks with heuristic algorithms to track the ball's trajectory. This informs critical batting decisions, taking into account the batsman's handedness and the ball's position on the field.

A significant achievement of this project is the successful implementation of real-time processing efficiency, achieved through automating the batting in cricket game.

1. Introduction

In the realm of sports gaming, EA Sports Cricket 07 stands as a notable title, captivating a broad audience among cricket-playing nations with its engaging gameplay. Our project, "Transforming Gameplay Through Vision Automation," aims to automate gameplay by integrating advanced vision automation techniques.

The existing vision automation techniques involve playing games such as Minecraft or board games. However, these games are slow-paced and involve Reinforcement learning or other similar techniques where the system has to make decisions with the motive of an end goal. The inspiration for our project stems from the intricate dynamics of cricket, where the precise trajectory and speed of the ball play pivotal roles in determining the game's outcome. This forms the motivation for us to automate the complex cricket game by utilizing low-level computer vision and Neural networks. Additionally, the existing work in this domain involves helping users by helping them detect

runes in the World of Warcraft or helping users mine farms in games such as Farmville. As seen all these techniques either aid users or do menial tasks, unlike the venture we have taken where we wanted to automate the entire batting and help our team win with the system we have built.

Our approach for automating EA Sports Cricket 07 includes the six key steps discussed in detail in the following sections. First, we detect the ball's position in each frame. Next, we track the ball's trajectory. Based on this tracking, we determine the optimal moment for a player to hit the ball. Additionally, our system can decide to leave the ball if it's tracked to be a wide ball, potentially gaining free runs.

The next section discusses the various steps we implement in our system.

2. Details of the approach:

2.1 Deciding the state of the Game:

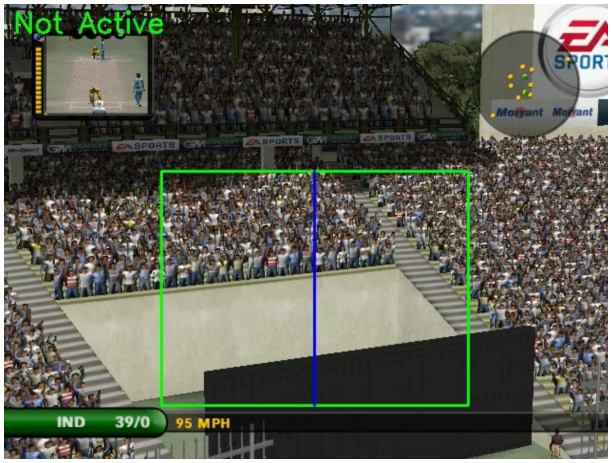


Fig 1: Showing "Not Active" State

In our project, we categorize game footage into three distinct categories: "Not Active," "Right," and "Left." The "Not Active" category encapsulates moments in the game that fall outside the immediate play, such as game replays, umpire decisions, and player reactions. The "Left" category is designated for moments when a left-handed batsman is at the strike, while the "Right" category to instances involving a right-handed batsman in action.

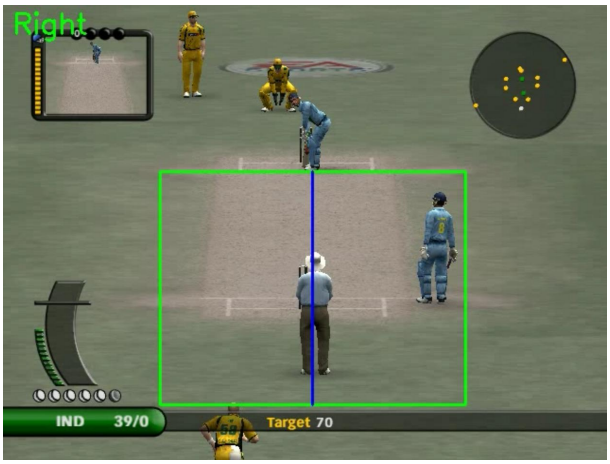


Fig 2: Showing "Right" State

To achieve this categorization, we recorded about 20 minutes of gameplay, extracting every frame for analysis. These frames provided the foundational data for our primary objective: training an ensemble neural network capable of accurately identifying the game status.

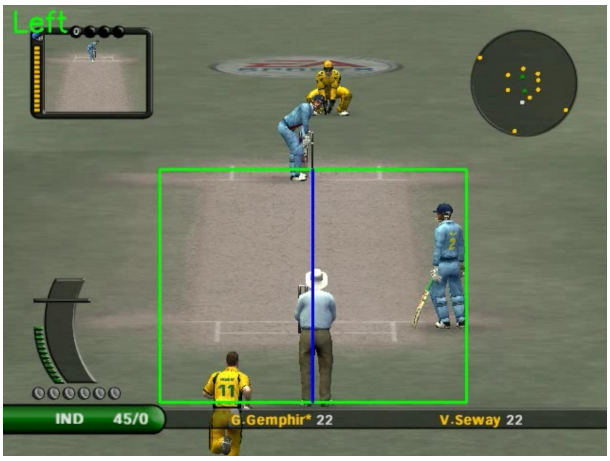


Fig 3: Showing "Left" State

After experimenting with various architectural designs for our model, we found that one particular configuration yielded superior results. The winning architecture of our Sequential model begins with a Conv2D layer with 32 filters, a kernel size of 3x3, and an activation function of 'relu', tailored to our input shape which includes the image width, height, and 3 color channels. Following this, we incorporated a MaxPooling2D layer with a pool size of 2x2 to reduce the spatial dimensions of the output. This pattern is repeated with another Conv2D layer, this time with 64 filters, and an accompanying MaxPooling2D layer. The model then flattens the data and proceeds through a

Dense layer with 64 neurons, again using 'relu' activation, culminating in a final Dense layer with 3 neurons corresponding to our categories, employing a 'softmax' activation function for classification.

```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(IMAGE_WIDTH,
IMAGE_HEIGHT, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(3, activation='softmax'))
```

Fig 4: Ensemble Neural Network Architecture

The training process involved processing 8985 images across these three classes, with an additional 2245 images for validation. Over the course of 10 epochs, our model showed substantial improvement and adaptation to the task. Notably, in the initial epoch, the model achieved an accuracy of 79.85% with a validation accuracy of 94.96%. This performance was refined over subsequent epochs, with epoch 3 marking a significant milestone: an accuracy of 99.24% and a validation accuracy of 95.00%. By the final epoch, our model achieved an impressive accuracy of 99.50%, although the validation accuracy slightly decreased to 92.77%.

This finely tuned architecture led us to achieve an overall accuracy rate of approximately 93%. Additionally, the "Left" and "Right" categories played a crucial role in determining whether a delivery was a wide ball, demonstrating the model's effectiveness in understanding nuanced aspects of the game. Our results indicate not only technical proficiency in frame categorization and ball tracking but also a strategic understanding of cricket gameplay.

2.2 Refining the Frame Focus Area for Enhanced Detection(Area of Interest):

A key decision in our design process involved narrowing our focus solely to the pitch area. This approach effectively reduced the frame size, significantly improving our ability to detect the ball within this smaller frame. The challenge, however, was determining the exact dimensions of this focus area. In cricket, the pitch size is a natural reference point, but the dynamic nature of the game camera necessitated a more adaptive strategy. After considerable effort, we developed a method to adjust the focus area. This was achieved by defining an imaginary rectangle, anchored by its top-left and bottom-right coordinates, which could be adjusted to align with the final position of the pitch, ensuring optimal coverage and accuracy in our analysis.

2.3 Detecting the Ball:

We implemented template matching to detect the ball in the game. This process involves taking snapshots of the ball from three different angles. For each frame, the size is determined by trackbars that define our area of interest. Within this area, we use the function `res = cv2.matchTemplate(rect_area, template1_gray, cv2.TM_CCOEFF_NORMED)`. This function returns a grayscale image, highlighting the degree of match between the neighboring pixels and the template.

After obtaining this image, we extract the maximum value using the command `min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)`. Template matching is then performed for each reference ball image initially captured, comparing them one by one. If the maximum value (max_val), which represents the confidence score of detecting the ball in a frame, exceeds 0.75, we proceed to additional operations, as outlined in subsequent steps.

The image below shows the result of the template-matching process. The central line is utilized by our system to determine the direction to hit the ball (left or right), which will be discussed later.

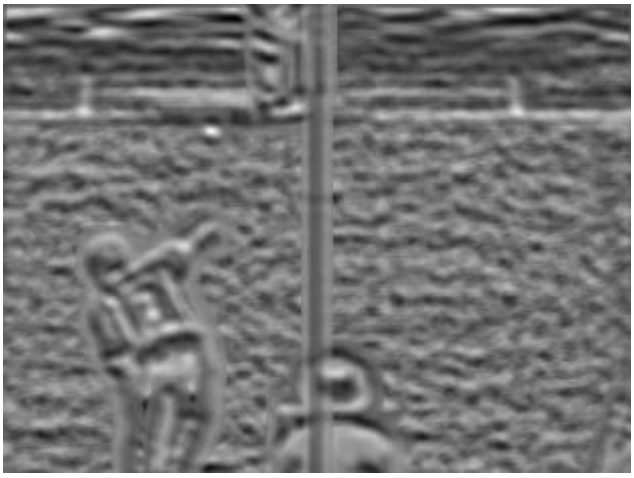


Fig 5: Template Matching

As shown in the estimate ball trajectory, our area of interest in only the frame that is drawn on the image. So considering that, in the frame we find the area where the ball is present as the white most or brightest part and this helps in the detection of the ball.

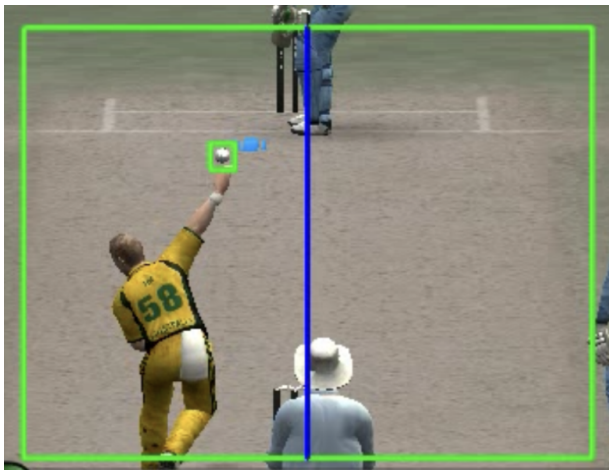


Fig 6: Ball detection

2.4 Tracking Ball:

In an effort to assist batsmen in making more informed decisions during play, we implemented a feature to track the ball's trajectory as it approaches them. This tracking is initiated when the ball is detected in the Area of Interest (AOI). We accomplished this by recording the central coordinates of the ball in consecutive frames and then plotting these points over the frames. This method effectively visualizes the trajectory of the ball. The resulting trajectory can be clearly observed in the following images, the blue line shows the trajectory of the ball.

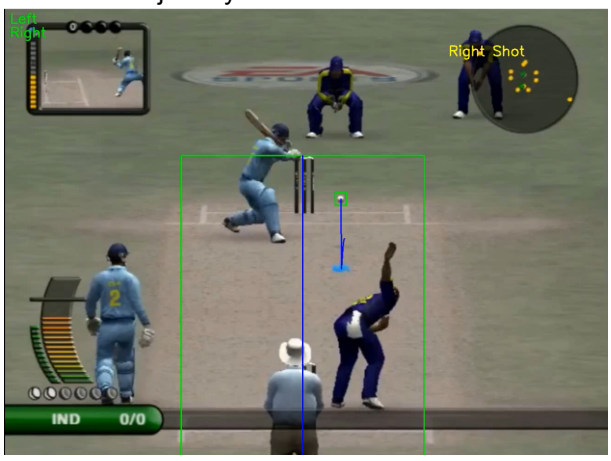


Fig 7: Ball Trajectory with Left-Hand Batsmen

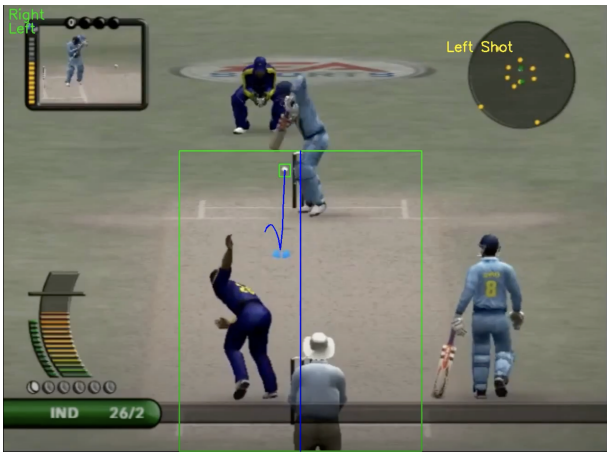


Fig 8: Ball Trajectory with Right-Hand Batsmen

2.5 Decision to hit the ball:

After successfully mapping the ball's trajectory, we leveraged the most recent position of the ball, coupled with the batsman's orientation (left or right-handed), to develop a heuristic method for deciding whether to play or leave the ball.

This decision-making process is activated only when the ball is within a reachable distance from the bat (top 40% of AOI). Importantly, the direction in which the batsman chooses to hit the ball is influenced by the ball's trajectory. This approach allows for a more strategic and responsive play, tailored to the dynamic conditions of each delivery.

```
def determine_shot(ball_x_center, middle_x, x1, x2,
count, handedness):
    if ball_x_center < middle_x and ball_x_center >
x1 + (middle_x - x1) * 0.70:
        return  ["s", "down"]
    if ball_x_center < middle_x and ball_x_center >
x1 + (middle_x - x1) * 0.50:
        if handedness == "Left":
            return  ["s", "down", "left"]
        else:
            return  ["s", "down"]
    if ball_x_center < middle_x:
        return  ["s", "right"]
    elif ball_x_center > middle_x and ball_x_center
< middle_x + (x2 - middle_x) * 0.70:
        return  ["s", "down"]
    elif ball_x_center > middle_x and ball_x_center
< middle_x + (x2 - middle_x) * 0.50:
        if handedness == "Right":
            return  ["s", "down", "right"]
        else:
            return  ["s", "down"]
    elif ball_x_center > middle_x:
        return  ["s", "left"]
```

2.6 Integrating the decisions into the game:

Upon finalizing the decision through our heuristic approach, the next step involved translating this decision into actual game controls for execution. To achieve this, we utilized the 'pyautogui' package. This package enabled us to programmatically simulate key presses corresponding to the game's control scheme. The action of hitting the ball in the game involves a combination of key presses, including *'s'* for a basic shot, the *arrow keys ('left', 'right', and 'down')* for directional shots, and the *'shift' key* for executing lofted shots.

By mapping our decision-making process to these specific key inputs, we were able to seamlessly integrate our heuristic approach into the game's control system, allowing for a more automated gameplay. For every shot executed in the game, we displayed the specific keys being pressed on the screen.

This visual feedback was not only useful for understanding the in-game actions being taken but also proved invaluable during our debugging process. It allowed us to track and verify the accuracy of the key presses being simulated by our system in real-time.

A sample of this can be seen in the following image, where the keys pressed for a particular shot are clearly displayed, providing insight into the decision-making and execution process of our automated gameplay.

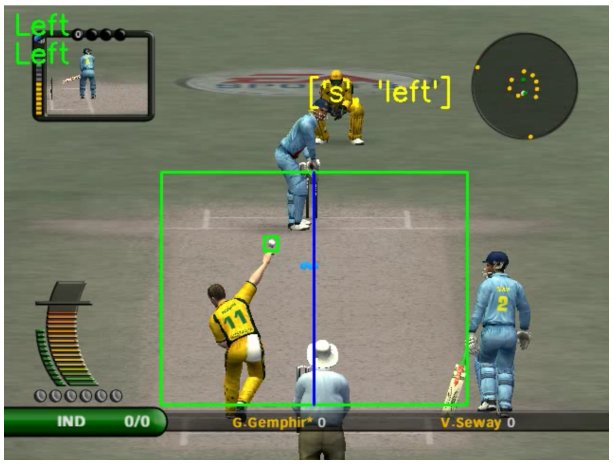


Fig 9: Displaying Key Strokes

3. Results:

The results from the simulated games clearly indicate that our system can effectively handle the complexities of a cricket game, making strategic decisions that result in successful gameplay. The ability of the system to score runs through boundaries highlights its advanced understanding of the game dynamics and its effective application of the developed algorithms. This success in actual gameplay scenarios validates the potential of our system in transforming how cricket games can be automated.

Link to our Results [Game videos]

3.1 Effectiveness in Gameplay:

A pivotal measure of our project's success was its performance in actual gameplay scenarios. Our automated system demonstrated a remarkable ability to compete effectively against opposing teams. This was evidenced by its proficiency in scoring runs, predominantly in the form of boundaries. By accurately categorizing frames and tracking the ball's trajectory, the system consistently made informed decisions, resulting in a high rate of successful shots.

3.2 Scoring Efficiency:

The efficiency of our system driven batsman was particularly notable in its ability to capitalize on scoring opportunities. The system's decision-making algorithm, combined with the real-time ball tracking, allowed it to discern when to play aggressive shots, resulting in an impressive tally of boundaries. This scoring efficiency was a direct result of our meticulously designed neural network and heuristic approach.

3.3 Performance Against Opposition:

Fig 10: Result versus South Africa

In simulated games against top-tier international teams like Australia, South Africa, and England, our automated system exceeded anticipated performance levels. It consistently amassed significant scores, demonstrating not only precision in ball detection and trajectory forecasting but also strategic excellence in choosing and executing shots. There are instances where it reached the target without losing a single wicket. Furthermore, when facing a less skilled bowler, our system empowered the batsman to achieve an impressive feat of scoring 36 runs in a single over, achieving this through hitting boundaries on every ball.

Fig 11: Result versus Australia

4. Discussion and conclusions:

In conclusion, our project has successfully demonstrated the potential of integrating low-level computer vision techniques with neural networks and heuristic algorithms to automate decision-making processes in dynamic environments, such as a cricket game. The high accuracy achieved in frame categorization, coupled with effective ball trajectory tracking and decision-making, highlights the effectiveness of our approach. Furthermore, our system's capability to translate these decisions into actual game controls represents a significant advancement in the integration of AI into real-time gaming and sports simulations.

4.1 Perspectives on Future Work:

Looking ahead, the potential for advancements in this field is immense. A key area for future development is enhancing the processing speed of our algorithm, aiming to accelerate the decision-making process without sacrificing accuracy. Future enhancements could include the ability to execute runs, and to play against spin and swing bowling. We categorize these as future work because accurately predicting ball trajectory in such complex scenarios requires deep learning techniques, which are not feasible with traditional computer vision algorithms.

Additionally, we could explore advanced object detection techniques, for improved ball detection in complex scenarios. One limitation of our current approach is the potential for missed detections, especially when the ball is thrown at a steep angle and then drops sharply. Given that our detection is confined to a designated area of interest, there are cases where the ball might not be detected, leading to missed shots. Implementing deep learning-based object detection methods could provide more accurate ball detection. However, these techniques require significant computing power, which may be beyond the capacity of standard laptops. For now, our geometric low-level computer vision techniques offer a fast and satisfactory solution within the constraints of our current computing resources.

Furthermore, expanding our system to accommodate various sports scenarios would not only increase the applicability of our technology but also provide valuable insights into the generalization capabilities of AI in sports.

5. Statement of individual contribution:

We utilized Google Drive and Google Colab for the sharing and maintenance of our code and

datasets. Our team collaboration was divided into three primary roles:

Uday Kanth Reddy and Girija Manoj Kumar Reddy: Focused on detecting the ball and estimating its trajectory. This crucial task involved pinpointing the ball's location in each frame and charting its course throughout the game.

Girija Manoj Kumar Reddy and Santosh Chejarla: Responsible for developing and refining the decision-making algorithm, particularly optimizing it for real-time processing. Their work was vital in determining the most appropriate actions based on the ball's trajectory and position.

Santosh Chejarla and Uday Kanth Reddy: Handled the input simulation, data collection, and the integration of these processes into the game. This included using pyautogui to simulate key presses and managing the data flow from the game to our decision-making algorithm and vice versa.

Through this team effort, using cloud-based collaboration tools, we were able to efficiently divide and conquer the various aspects of our project, leading to a seamless integration of all its components.

6. References:

1. https://www.ea.com/en-gb/games/cricket/cricket-2007
2. https://en.wikipedia.org/wiki/Cricket
3. https://pysource.com/2021/11/02/kalman-filter-predict-the-trajectory-of-an-object/
4. https://docs.opencv.org/4.x/d4/dc6/tutorial_py_template_matching.html
5. https://pjreddie.com/darknet/yolo
6. https://buddhaman.itch.io/football-evo
7. **Link to Our Data, Output Videos and Code**