A-Eye: Automating The Role Of The Third Umpire In The Game Of Cricket

Dr. Tariq Mahmood
Department of Computer Science
National University of Computer and Emerging Sciences
Karachi, Pakistan
Email: tariq.mahmood@nu.edu.pk

Syed Obaid Ahmed
Department of Computer Science
National University of Computer and Emerging Sciences
Karachi, Pakistan
Email: k050313@nu.edu.pk

Muhammad Hadi Swaleh
Department of Computer Science
National University of Computer and Emerging Sciences
Karachi, Pakistan
Email: k050218@nu.edu.pk

Syed Hassan Nayyer
Department of Computer Science
National University of Computer and Emerging Sciences
Karachi, Pakistan
Email: k060188@nu.edu.pk

Abstract—Cricket is a popular sport that involves two teams, say Team A and Team B. Each team comprises eleven players, along with two field umpires. Based on the outcome of a toss, one of the teams, say Team A, initially bats (by using a wooden structure called a bat) in order to score some runs. These runs are scored while Team B is balling (by using a special type of a ball). Then, the roles get reversed: Team A balls and Team B bats, while trying to overcome the score posted by Team A. If this happens, then Team B wins. Otherwise, Team A wins. The balling team aims to get the batsmen out, i.e., dismiss them from batting, in order to minimize the scoring. One of the ways in which this could happen is a run-out, i.e., the batsman fails to enter a particular playing area, before the bowling team dislodges three wooden stumps in that area. It is generally very difficult for the umpires to detect this scenario through the naked eye. Hence, it is typically referred to a third umpire, who makes the ‘Run-Out/Not-Out’ decision through video technology. This process typically consumes 30 seconds (or more) which disrupts the fast pace of the game, e.g., in Twenty20 cricket (a type of cricket game). In this paper, we have proposed and implemented a novel technology for cricket, called A-Eye, which automates the role of the third umpire. By applying A-Eye to a set of autonomously-filmed run-out videos, we have shown that it is extremely efficient as compared to the third umpire, and almost as accurate. Also, it has the potential to minimize decision errors made by third umpires, and is able to estimate a rating for the field umpires. These results have convinced our local cricket council to employ A-Eye within a professional cricket tournament, which will be held soon.

I. RELEVANT BACKGROUND AND MOTIVATION

Cricket is a sport that is currently being played across 105 different countries. It is played within an oval-shaped arena called a cricket field. The focus of the play is the pitch, a 22-yard (20.12 meters) rectangular area located at the center of the field. Figure 1 shows a part of the cricket field and its pitch. At both ends of the pitch, there are two regions marked by white lines in a similar way. These are called creases. In the middle of each crease, three wooden stumps are planted. These are collectively called the wicket (see Figure 1). In cricket, two major activities occur on the pitch: 1) batting, and 2) bowling. Batting is done through a special piece of wooden equipment called a cricket bat, while bowling is done through a special ball called the cricket ball (henceforth labeled as ball). Figure 2 shows a cricket bat and ball. In any cricket game, two teams compete against each other, each comprising eleven players. We assume that these teams are labeled as Team A and Team B. We also assume that initially, Team B bowls, while Team A bats in order to score a maximum number of runs (numerical counts). Then, the roles get reversed: Team A bowls, while Team B bats in order to overcome the score of Team A.
and hence, win the match. During a game, the bowling team tries to stem the scoring rate of the batting team, notably by disqualifying each player from batting (one-by-one). When this happens, we say that the player is out. This decision is typically given by a cricketing umpire. In any game, there are two umpires on the field, known as “field umpires”. Figure 3 shows one of these in a black and white dress.

We will now formalize these concepts in the context of this paper. Any player who bowls is called the bowler, while a player who bats is called the batsman. From a given crease, the bowler delivers (throws in a particular way) the ball to the batsman, who stands at the other crease. The batsman’s job is to try and hit the ball to some area of the field. For instance, Figure 3 shows a bowler who has just delivered his ball while running from one end. Also, the batsman, with his bat raised, is getting ready to hit the ball. The purpose of hitting the ball is to try and score runs. In this paper, we are concerned with a type of scoring called “running between wickets”. Specifically, the batting team has always two batsmen at the crease (as seen in Figure 3). If one of them hits the ball far enough, then these two can cross over, i.e., exchange positions by going from one crease (wicket) to the other one. One such exchange leads to the addition of a single run. It is important to note that, for a run to be valid, each batsman is required to place his bat within the crease (on the ground) once he reaches the opposite end. For instance, Figure 4 shows the batsmen crossing over for one run, while Figure 5 shows a batsman trying to place his bat within the crease.

If a batsman is unable to ground his bat in time, then he can get out. This type of dismissal is called a run-out. Specifically, after the ball has been hit, a member of the bowling team tries to stop the ball as soon as possible, and throws it to another member called the wicket keeper (henceforth labeled as keeper). This person typically stands behind the wicket (see Figures 5 and 6). As soon as he catches the ball, the keeper uses it in order to dislodge the wicket. If, at this particular time, the crossing-over batsman is still out of his crease, or has not grounded his bat, then he is run-out. For instance, Figure 6 shows a run-out scenario in which the wicket has been dislodged by the keeper, while the batsman is still out of his crease. The decision whether a batsman is run-out (or not) should be given by the field umpire. However, the problem is that the run-out activity occurs very quickly. So, mostly, it is not possible for the umpire’s naked eye to keep track of the movement of the bat, as well as that of the ball (at the wicket). In such a situation, the decision is referred to a third umpire, who sits outside the field. He watches the video of the run-out in slow motion, and hence, detects easily whether the batsman grounded his bat in time (i.e., he is ‘Not-Out’, Figure 7), or not (i.e., he is ‘Out’, Figure 8).

A. Problem Statement

While the third umpire is making his decision, all the players have to wait for it, and the game stops entirely. In fact, according to the standard cricketing rules, the third umpire should make his decision within 30 seconds. However, on the

1There are other ways in which a batsman could get out, but they are not within the scope of this paper.

2http://static.icc-cricket.yahoo.net/ugc/documents/
average, third umpires consume from 45 seconds to more than a minute\(^3\). This situation causes two major, negative effects on the outcome of the game:

1. It disrupts the playing rhythm of the players, i.e., the flow of the game, and
2. It leads to a loss of playing time for both the teams.

We now describe these effects in detail. In cricket, the goal of each team is to behave so as to maximize its chance of winning. To this end, a useful tactic typically adopted by a team is to try and disrupt the playing rhythm of the other team. For instance, sometimes, a batsman might deliberately take some time in preparing to face a delivery, which leads the bowler to wait, hence disrupting his bowling rhythm\(^4\). Such tactics are legal and definitely affect the outcome of the game. However, in the scenario of a referral to the third umpire, stopping the play disrupts the rhythm of both the batsmen and the bowlers unnecessarily. Hence, according to professional third umpires, the consequent outcome of the match might not be justified.\(^5\)

Such a scenario becomes very crucial in a type of cricket game called T20, in which both the teams face a limited number of deliveries (120), and try to score as much runs as possible at a very fast pace. In this situation, if a run-out occurs and there is a referral to the third umpire, the fast pace of the game is completely disrupted. So, players who were performing well before the referral might lose their rhythm after it. In fact, we have personally watched numerous videos of different T20 matches, in which referrals affected our expected outcomes of these matches\(^6\). For instance, we witnessed that several bowlers bowled brilliantly before a referral, by preventing the batsmen from scoring lots of runs. However, after the referral, the same bowlers lost their rhythm, hence leading to an increased scoring rate. Similarly, there were several batsmen who were scoring lots of runs before a referral, but after a referral, their scoring rate decreased, or they got out, due to a disruption of their rhythm. In almost all such scenarios, the disruptions completely changed our expectations, i.e., the team which we expected to win, lost in the end (and vice versa).

In this context, another limitation is that of playing time. In T20, the bowling team should deliver 120 deliveries within a stipulated amount of time. In case a referral takes a lot of time, this leads to a loss of playing time. Consequently, the field umpires reduce the number of balls that must be delivered, e.g., from 120 to 100. Doing so often invalidates the playing strategies of the teams. Then, they must re-design their plans of actions in a very limited amount of time, which don’t always turn out to be effective. Hence, the consequent outcome of the match might get affected negatively for one of the teams.

Moreover, third umpires are quite fallible, i.e., they can give wrong Run-Out/Not-Out decisions due to human error, e.g., in a match between the teams of India and South Africa\(^7\), and in a match between India and Pakistan\(^8\). In fact, such decisions could infuriate the players considerably, hence further affecting their playing rhythm.\(^9\)

From the aforementioned limitations, it is obvious that we need to minimize the decision time of the third umpire, and to substitute it by a more robust (error-free) alternative. \textit{In this paper, we address this challenge by automating the role of the third umpire.} Specifically, we have designed and implemented a novel technology, labeled as Artificial Eye (A-Eye), which exploits image processing techniques in order to automatically output the decision (Run-Out or Not-Out), given a run-out video as an input. We have validated our application with run-out videos that we filmed autonomously, in order to cater for a particular requirement related to the position of the camera. We have proved that A-Eye is extremely efficient as compared to the third umpire, and is almost as accurate. A-Eye has the potential to minimize the errors made by a third umpire, and it can also be used to calculate a rating for the field umpires. In fact, automation techniques are being currently employed in order to support the field umpires in giving ‘Out/Not-Out’ decisions, for situations other than a run-out, e.g., the HawkEye feature [2] for Leg-Before-Wicket

\(^3\)http://news.oneindia.in/2008/07/22/time-consuming-referral-system-to-alter-test-results-says-harihar-1216703385.html

\(^4\)http://www.guardian.co.uk/sport/blog/2010/jun/11/jonathan-trott-scratch-england-bangladesh

\(^5\)http://www.espncricinfo.com/action-t20/content/story/140077.html

\(^6\)http://www.thefreelibrary.com/LAW+OF+THE+BUNGLE;+HOLLIOAKE+FURY+AT+THIRD+UMPIRE’S+RUN-OUT+ERROR.-a0110687962
(LBW) decisions [1]. However, no state-of-the-art work exists for automating the run-out decisions. Hence, our technology is quite novel, and is of extreme practical value for all types of cricket matches.

This paper is structured as follows. In Section II, we will describe and illustrate the working of various architectural components of A-Eye in details. Within this section, we will also describe the particular algorithm for automating the run-out decision. Later on, in Section III, we will compare the performance of A-Eye with a third umpire and explain our results. Finally, in Section IV, we will present our conclusions along with the future work.

II. SYSTEM ARCHITECTURE

We have implemented A-Eye as a desktop application in the C-Sharp(\#) programming language. It’s architecture is shown in Figure 9. Here, we have adopted a three-tier approach, i.e., the client tier, the middle tier and the back-end tier. In the client tier, the users interact with two different Graphical User Interfaces (GUI’s) of A-Eye, i.e., GUI 1 (Figure 10) and GUI 2 (Figure 11). Given a run-out video, GUI 1 is initially used to load and perform some pre-processing tasks, and GUI 2 is then used to detect the motion at the wicket and the crease, and hence, to detect whether the batsman has been run-out (or not). These activities are performed in the middle tier, which consists of eight different modules. The back-end tier comprises an image (video) repository as well as a DBMS, which are employed by the middle tier during run-out detection. We will now describe the eight modules of the middle tier, which form the crux of A-Eye’s functionality.

A. Process Video Module

In the “Process Video” module, we have implemented a complete video player within GUI 1. It is located in the top-left corner in GUI 1. It allows users to perform two video-related operations: 1) load a run-out video, and 2) check whether it is able to run smoothly, i.e., it does not hang during play. If the video does hang, then image processing techniques cannot be effectively applied for run-out detection [3]. Figure 12 shows that a video has been loaded in GUI 1, and it is being tested for smooth running. The current video snapshot shows that the batsman has just placed his bat in the crease, and at the same time, the ball has touched the wickets.

B. Split Video Module

Once a video has been tested, it is sent to the “Split Video” module, which divides the video into frames, i.e., a set of stationary images that constitute the video. This module is

\textsuperscript{10}It supports all video playing formats, e.g., MPEG, AVI etc.
located in the bottom-left corner in GUI 1. It is required because traditional image processing techniques are applied on still images (and not directly on streaming video) [3]. In our case, we are interested in detecting the motion of both the bat and the ball in the same given frame (described in Section II-D). The “Split Video” module divides the video in Figure 12 into 108 frames; in Figure 13 we have selected the frame numbered 96. In order to facilitate image processing, we verify all frames separately in order to ensure that none of them are corrupted or unclear [3].

C. Gray Scale Converter

Once a video has been divided into frames, our next step is to detect two major objects within a frame, i.e., the crease and the wicket. This is necessary because the activity (of the bat and ball) occurring around these objects will assist us in automating the run-out (described below). For this, we initially need to perform a pre-processing technique called grayscaling. Specifically, videos typically exist as analog (colored) signals, and it is necessary to convert them into a digital signal in order to effectively apply image processing techniques, e.g., object detection [3]. Typically, there exist two such types of conversions: 1) one in which a frame is converted into a binary format, i.e., into a black and white color, and 2) one in which a frame is converted into a discrete numbers of shades of gray, i.e., into the grayscale mode. For our application, we convert the selected frame into grayscale, because it generally supports a more accurate detection of objects as compared to a black-and-white image [3]. This is done by the “Gray Scale Converter” module, which is located in the bottom-right corner of GUI 1. For instance, the frame in Figure 13 has been converted into grayscale in Figure 14. Here, we have shown only one of the converted frames, but we actually convert all the video frames into grayscale, and store them.

D. Motion Detection Algorithm

In order to describe the remaining modules, it is now necessary to describe our algorithm for run-out detection. Specifically, in order to detect a run-out, we need to detect the motion of the ball (or the wicket keeper’s hand) as it approaches the wicket, and the motion of the bat as it approaches the crease. Moreover, as these two motions occur simultaneously, so they must be detected simultaneously as well. For this, we employ traditional image processing techniques which detect objects in motion, i.e., they output those objects which are actually moving across a given set of consecutive frames. In our application, we employed a Motion Detection Algorithm (MDA) that is based on a simple comparison of the pixels across consecutive frames. For instance, assume that frame A and frame B are two consecutive frames of some video. Also assume that in frame A, a set of pixels (making up a moving object) are different from the same set of pixels in frame B. Then, we can conclude that frame B is different from frame A because the object has moved. We label this difference as the frame difference, and we set its threshold to 0.1, i.e., we conclude that we have detected motion if there is a 10% difference between the frames (described in Section II-H). It is important to mention that the frame difference technique
works only if the background (or the surrounding area of the moving object) is constant, i.e., it remains perfectly unchanged across the consecutive frames. In case this doesn’t happen (e.g., due to noise), then we would also detect insignificant motions (objects). Once we have identified the motion regions in a frame, we use a technique known as **blob counting** [3], in order to fit a red-colored rectangle around each region. This allows us to determine the amount of detected objects, as well as the position and size of each detected object.

**E. Performance of MDA**

In order to test the performance of MDA on the wicket and crease markers, we tested it on a set of 20 run-out videos from the image repository. We label this set as the ‘MDA Set’. We filmed some of these videos ourselves, while the remaining ones were downloaded from the Internet. We converted each video into frames, and then into grayscale in GUI 1. Then, for each video, we selected a set of frames which were relevant to the run-out, i.e., in which the bat was visible and very near to the crease, and the ball (or wicket keeper’s hand) was near to the wicket (for instance, see the frame in Figure 13). We label this set of frames as the **relevant frame set**. We determined the relevant frame set for each selected video, and for each set, we tried to detect the simultaneous motion near the crease and the wicket using MDA. We show our results for three videos, in Figures 15, 16, and 17. We selected these videos because they represent the complete set of our results (listed below). Figure 15 shows a relevant frame from a T20 match, in which A-Eye has detected four objects in motion, i.e., the player, the bat, the ball, and the wicket. Moreover, Figure 16 shows a frame from the video in Figure 12, in which A-Eye has detected five objects in motion: the player with the bat, a player in the background, and three separate parts of wicket. Finally, Figure 17 shows a frame in which A-Eye has detected three objects in motion: the player with the bat, the wicket, a moving car in the background. These images allow us to make the following conclusions (marked as ‘C’ below) regarding run-out detection:

1. **C1:** MDA is always able to detect the motion of the wicket. This happens because a wicket is an object of considerable size, and the frame difference method is always able to detect large-sized objects [3].
2. **C2:** MDA can detect more than one motion region at the wicket (Figures 16 and 17). Such a scenario can lead to an indecision regarding the detection of a run-out, i.e., which region we should use for comparison with the motion region of the bat? A reasonable solution is to select a particular part of the wicket for each run-out video, and to use only the motion region associated with this part. For instance, we could use the top-most part because it is the one most commonly used by wicket keepers in order to run-out a batsman (see Figure 6).
3. **C3:** MDA doesn’t always detect the motion of the ball, which maybe due to the ball’s small size. This is not a problem for our task because it is the ball itself which causes the motion of the wicket, and we can always detect the wicket’s motion (C1).
4. **C4:** MDA detects insignificant objects that are not relevant for run-out detection. This is due to noise in the background, e.g., a player and a moving car in Figures 16 and 17 respectively. In fact, a run-out video might not have a constant background, i.e., there can be motion either due to the audience (Figure 4), or due to the

---

11In GUI 2, we apply the MDA on relevant frames that have been converted into grayscale. However, for the sake of interactivity, we only show the (original) colored version of these gray-scaled frames.
players themselves (Figure 6). A reasonable solution for minimizing this noise is to mark areas in the relevant frames for detecting the wicket and the crease. Then, we can ignore the (noisy) objects whose motion doesn’t occur around these areas.

- **C5:** MDA is not always able to detect the motion of the bat, and it often considers the player and the bat as one moving object (Figures 16 and 17). In fact, the motion of the bat can be different from the motion of the batsman. For instance, in Figure 6, the batsman’s motion might be slower as compared to his bat’s motion, which he is trying to ground in the crease. So, in principle, both these motions should be detected separately. However, our results show that even detecting them together doesn’t affect the automated run-out decision in any way (proved from results shown in Section III).

- **C6:** MDA is never able to detect the crease. In fact, in any relevant frame, the only indication that a crease is an ‘object’ is the white line that is located at a straight-line distance of 1.22 meters from the wicket (see Figure 1). MDA never detects this line because its size is quite insignificant. This creates a problem: how can we detect motion at the crease? A reasonable solution is to autonomously mark an area representing the crease, for each relevant frame. Then, MDA can detect the bat’s motion in this area only.

As concluded in C2, C4 and C6, we need to fix two areas within each relevant frame, in order to facilitate motion detection at the crease and wicket. Hence, in GUI 2, we exploit two identification markers: 1) the crease marker, which is the horizontal line in the center for motion detection at the crease, and 2) the wicket marker, which is the vertical black square located in the top-right corner for motion detection at the top-most part of the wicket. These markers are automatically applied to any frame loaded in GUI 2. Figure 11 shows the default positions of these markers, and Figure 18 shows these markers being applied to the frame selected in Figure 13.

### F. Object Tuner and Object Detector Modules

We now resume our discussion of the modules of A-Eye (Figure 9). We allow the user to ‘tune’ (adjust) the position of the crease and wicket markers, for a given relevant frame. This can be done in two dimensions, i.e., vertically (over the y-axis) and horizontally (over the x-axis). This tuning process is performed by the “Object Tuner” module of A-Eye. After tuning, the “Object Detector” module exploits the MDA in order to initially detect the wicket marker and the crease marker, which are our ‘objects’ for run-out automation. In fact, for each video in the MDA Set, this module was able to successfully detect the wicket marker, i.e., we obtained a red rectangle around the wicket marker (as described above). However, it didn’t always detect the crease marker. This latter result occurred in those relevant frames in which the camera was positioned at a particular height from the ground (for instance, see Figures 15 and 6). In this situation, we are trying to detect motion at a (horizontal) line (crease marker) in a 3D coordinate frame. However, traditional techniques for line detection are based on a 2D coordinate frame [4], [5], [3], [6]. As these techniques have been validated in a diverse number of applications [6], [4], we will exploit one of them, called the Hough Transform for our task (described below). In fact, novel techniques do exist for line detection in 3D, but most of them are based on the 2D techniques (for instance, refer to [7], [8], [9], [10]). Also, they have not been standardized (as compared to 2D techniques) and are the domain of extensive research these days.

We will now describe the Hough transform. In order to detect the crease marker, we first apply edge detection techniques [3], to obtain a limited number of pixels that actually lie on the crease marker. The idea is to detect the crease marker by using regression techniques on these pixels. However, due to a noisy background, all such pixels are not accurately obtained. So, the crease marker cannot be detected. To solve this problem, we initially consider the equation of the crease marker, i.e., \( y = m \cdot x + b \) where \( m \) is the gradient and \( b \) is the y-axis intercept. Typically, this allows us to plot each point \((x, y)\) in 2D. Using the Hough transform, we parameterize the 2D space by plotting each point as \((r, \theta)\). Here, \( r \) represents the distance between the crease marker and the origin, while \( \theta \) is the angle of the vector from the origin to this point (see Figure 19). Note that in GUI 2, we have set the origin \((0, 0)\)
as the bottom-left corner. We can write the equation of the crease marker as:

$$y = \left(\frac{\cos \theta}{\sin \theta}\right)x + \left(\frac{r}{\sin \theta}\right)$$  \hspace{1cm} (1)

which is re-written as

$$r = x\cos \theta + y\sin \theta$$  \hspace{1cm} (2)

Thus, each point \((r, \theta)\) is unique if \(\theta \in [0, \pi)\) and \(r \in \mathbb{R}^{12}\), or if \(\theta \in [0, 2\pi)\) and \(r \geq 0\). If a pixel on the crease marker has coordinate \((x, y)\) in the initial 2D space, then all possible lines that can pass through this pixel obey Equation 2 in the parameterized space. In our case, the only possible line is the crease marker. In fact, Equation 2 corresponds to a sinusoidal curve in the \((r, \theta)\) space, which is unique to this point. If the curves corresponding to two such points (on the crease marker) are superimposed, then the location where they cross correspond to lines (in the original image space) that pass through both points. More generally, even if we are able to detect a limited number of pixels with edge detection, we can generate curves between these pixels in the parameterized space, and superimpose these curves in order to detect the crease marker. Once this is detected, any motion of the bat near this marker can be easily detected.

Employing the Hough transform necessitates that the run-out videos should be filmed in 2D, i.e., the camera should be located on the ground (at zero height). In fact, we have autonomously filmed 30 such run-out videos in order to successfully test the performance of A-Eye (refer to Section III), two of which are shown in Figures 16 and 17. This simple requirement allows our A-Eye technology to be easily utilized during play, by installing a (hidden) camera that is located on the surface of the ground (zero height) and facing the wicket.

G. Pixel Capture Module

Once both the markers have been detected, the “Pixel Capture” module captures all the pixels related to these markers. Specifically, for each frame in the relevant frame set, it captures the 50 pixels that comprise the wicket marker. Also, for the crease marker, it uses three pre-defined rectangles of equal size, where each rectangle comprises 600 pixels. In Figure 20, these rectangles have been labeled as A, B and C. The movement of bat (starting from A) can be captured by the pixels comprising these three rectangles. For instance, if there are 6 frames in a relevant frame set, then A, B and C can comprise the bat’s motion for the first two frames, third and fourth frames, and the last two frames respectively.

H. Decision Detector Module

Once the pixels have been captured, the “Decision Detector” (DD) module detects a Run-Out (or a Not-Out) by comparing the content of the pixels. Let us represent the frame difference obtained at the wicket and crease markers as the boolean variables \(\text{WicketChange}\) and \(\text{CreaseChange}\) respectively. Then, \(\text{WicketChange}\) and \(\text{CreaseChange}\) are “true” if a 10% change occurs across consecutive frames at the wicket and crease respectively; otherwise, they are false. In this context, DD makes a decision based on one of the following four situations:

- \{\text{WicketChange}=true,\text{CreaseChange}=false\}, i.e., the wicket has been dislodged but the batsman is out of his crease. In this situation, DD decides that the batsman is ‘Run-Out’.
- \{\text{WicketChange}=false,\text{CreaseChange}=true\}, i.e., the wicket has not been dislodged and the batsman is in the crease. In this situation, DD decides that the batsman is ‘Not-Out’.
- \{\text{WicketChange}=true,\text{CreaseChange}=true\}, i.e., the wicket has been dislodged and the batsman is within the crease. In this situation, DD decides that the batsman is ‘Not-Out’. Specifically, the bat arrives in crease at the same time that the wicket was dislodged, and third umpires typically rule this as a ‘Not-Out’.
- \{\text{WicketChange}=false,\text{CreaseChange}=false\}, i.e., the wicket has not been dislodged and the batsman is out of his crease. In this situation, DD decides that the batsman is ‘Not-Out’.

For instance, Figure 21 depicts the situation \{\text{WicketChange}=false,\text{CreaseChange}=true\}. Here, the bat’s motion is detected at the crease marker in the selected frame, shown by the dialog box “Bat In The Crease Now”. However, the wicket has not been dislodged in this frame. Hence, the decision is “Not Out” (shown in Figure 23). In fact, the motion at the wicket marker is detected in the next (consecutive) frame, shown by the dialog box “Bails Off Now” in Figure 21\(^{14}\). This depicts the situation \{\text{WicketChange}=true,\text{CreaseChange}=true\}, for which the decision is again “Not Out” (Figure 23). Moreover, Figure 24 depicts the situation

\(^{12}\)\(R\) is the set of real numbers.

\(^{13}\)This situation occurs for a non-relevant frame, and a decision is provided here only for the sake of specification

\(^{14}\)“Bails” is another word for the top-most part of the wicket.
{WicketChange=true,CreaseChange=false}, in which the bat’s motion has not been detected in the selected frame (as it is in the air), but the wicket has been dislodged in the same frame. Hence, the decision is (Run)“Out”.

I. Umpire Rater Module

Automating run-out decisions means that A-Eye can be used to calculate a rating for the performance of the field umpires. The ratings are assigned by the “Umpire Rater” module. Specifically, if the field umpire makes a run-out decision by himself, he uses his naked eye to detect the motion of the bat and the ball. This scenario is depicted in Figure 25. Here, each X-coordinate represents one frame of a run-out video, and the y-axis represents the time taken to detect an even in a frame. Situation ‘A’ represents the event “Bat In The Crease Now” (Figure 21), and situation ‘B’ represents the event “Bails Off Now” (Figure 22). So, Situation ‘C’ represents the number of frames elapsed between the detection of A and B. We set a threshold for C, i.e., 5 frames. Then, the following two situations can occur:

- $C \leq 5$: Here, we assume that detecting A and B is quite tough for the field umpire, as they occur in very quick succession. So, if he is still able to give the correct run-out decision, then we increase his rating by a certain percentage, i.e., $rating_{Up}$.
- $C > 5$: Here, we assume that enough frames have elapsed in order to allow the field umpire to make the run-out decision autonomously. So, if he still refers the decision to the third umpire, we reduce his previous rating by a certain percentage, i.e., $rating_{Down}$.

In order to determine reasonable values for $rating_{Up}$ and $rating_{Down}$, we are currently analyzing the process adopted by different cricketing councils in order to assign ratings to umpires, e.g., the Elite Panel Umpire rating assigned by the
International Cricket Council (ICC)\(^{15}\).

III. RESULTS

In order to validate the performance of A-Eye, we compared it with the performance of the third umpire. For our experiment, we personally filmed a set of 30 run-out videos in 2D (with camera at zero height). We label this as the “Test Set”. In this set, the batsman was ‘out’ 21 times, and he was ‘not-out’ 9 times. We deliberately simulated these decisions, i.e., we played in such a way to ensure a ‘Run-Out’/‘Not-Out’ situation. This allowed us to set these decisions as our benchmark, and we label them as ‘benchmark decisions’. Then, we took 5 samples, each of 15 videos, from the Test Set. This was done to provide more diversity in the ‘Run-Out’/‘Not-Out’ distributions, and hence, more flexibility in the results [11]. This distribution for each sample is shown in Table III. Here, the terms ‘Positive’ and ‘Negative’ in the columns specify the number of videos in which the batsman was ‘Run-Out’ and ‘Not-Out’ respectively. These terms will be used to present results related to run-out accuracy (Section III-A). We tested each video in each sample with A-Eye, as well as with a third umpire. The latter was a neutral individual who made run-out decisions similarly to professional third umpires. We compared the run-out decisions of A-Eye and the third umpire along two dimensions: 1) run-out accuracy, i.e., how much similar are the decisions of A-Eye/third umpire to the corresponding benchmark decision, and 2) run-out time, i.e., the time taken by A-Eye/third umpire to make these decisions. We use the labels ‘A-Eye’ and ‘3Ump’ to refer to the results from A-Eye and the third umpire respectively.

A. Run-Out Accuracy

In this section, we compare the run-out accuracy of A-Eye with 3Ump. To this end, we model the run-out decisions of the Test Set as a binary classification problem [12]. Specifically, we assign the class labels ‘Positive’ and ‘Negative’ to the decisions ‘Run-Out’ and ‘Not-Out’ respectively (as mentioned previously). Then, for each sample in Table III, we compute the confusion matrix [12], which comprises the following four entities:

1) True Positive (TP), i.e., A-Eye/3Ump decides ‘Run-Out’ for a Positive video,

2) False Negative (FN), i.e., A-Eye/3Ump decides ‘Not-Out’ for a Positive video,

3) False Positive (FP), i.e., A-Eye/3Ump decides ‘Run-Out’ for a Negative video, and

4) True Negative (TN), i.e., A-Eye/3Ump decides ‘Not-Out’ for a Negative video.

These computed values for these entities are shown in Table III-A. In order to understand these values, we compute the True Positive Rate (TPR) as

\[
TPR = TP / (TP + FN)
\]  

(3)

and the False Positive Rate (FPR) as

\[
FPR = FP / (FP + TN)
\]  

(4)

For both A-Eye and 3Ump, we compute the TPR and FPR values for each sample. In Figure 26, the green boxes and blue crosses represent these values for A-Eye and 3Ump respectively. We also estimate the ROC curves for A-Eye and 3Ump\(^{16}\). The red diagonal line represents the decision boundary; any point above this line represents an accurate run-out detection, while a point below this line represents an inaccurate one. So, our first result is that A-Eye is able to automate the run-out decision accurately for each sample. In fact, points (samples) that are close to the upper-left corner represent the most accurate decisions, and the coordinate \(\{0, 1\}\) represents 100% accuracy. In Figure 26, we can see that A-Eye has detected 4 samples very accurately and the remaining sample perfectly. The major result is that the accuracy statistics for A-Eye are very similar to that of 3Ump, i.e., the third umpire has also detected 4 samples very accurately and the remaining sample perfectly. However, these 4 detections are a bit more accurate (closer to \(\{0, 1\}\)) as compared to those for A-Eye. This can be seen from the fact that the ROC curve for 3Ump is higher than that for A-Eye. Such a behavior is normal, considering that the third umpire consumes considerably more time as compared to A-Eye (refer to Section III-B). Hence, we

<table>
<thead>
<tr>
<th>Sample No.</th>
<th>Positive</th>
<th>Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>S2</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>S3</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>S4</td>
<td>11</td>
<td>4</td>
</tr>
<tr>
<td>S5</td>
<td>12</td>
<td>3</td>
</tr>
</tbody>
</table>

**Table I**

**Run-Out/Not-Out Distribution For Five Samples Of The Test Set; Positive=Run-Out; Negative=Not-Out**

<table>
<thead>
<tr>
<th>Sample</th>
<th>TP</th>
<th>FN</th>
<th>FP</th>
<th>TN</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1_AEye</td>
<td>12</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>S1_3Ump</td>
<td>11</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>S2_AEye</td>
<td>12</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S2_3Ump</td>
<td>12</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>S3_AEye</td>
<td>9</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>S3_3Ump</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>S4_AEye</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>S4_3Ump</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>S5_AEye</td>
<td>10</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>S5_3Ump</td>
<td>11</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

**Table II**

**TP, FN, FP, AND TN VALUES FOR BOTH A-EYE AND 3UMP; SAMPLE ‘N’_A-EYE AND SAMPLE ‘N’_3Ump DENOTES THESE VALUES FOR SAMPLE 1 WHEN A-EYE AND THE THIRD UMPIRE WAS EMPLOYED FOR RUN-OUT DETECTION, RESPECTIVELY**
Typically, such situations require a decision by an off-the-field umpire whether a batsman is out (or not-out) in a run-out situation. That, for each sample, in Table III. Here, columns T_3Ump denote these times for A-Eye and the third umpire respectively; column % Red. represents how much percent A-Eye is faster than the third umpire.

### Table III

<table>
<thead>
<tr>
<th>Sample</th>
<th>T_AEye (sec)</th>
<th>T_3Ump (sec)</th>
<th>% Red.</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>0.43</td>
<td>45.4</td>
<td>99.0</td>
</tr>
<tr>
<td>S2</td>
<td>0.53</td>
<td>35.9</td>
<td>98.5</td>
</tr>
<tr>
<td>S3</td>
<td>1.1</td>
<td>66.3</td>
<td>98.3</td>
</tr>
<tr>
<td>S4</td>
<td>0.9</td>
<td>23.5</td>
<td>96.2</td>
</tr>
<tr>
<td>S5</td>
<td>1.6</td>
<td>55.3</td>
<td>97.1</td>
</tr>
</tbody>
</table>

**Run-Out Times (in seconds).** Columns T_AEye and T_3Ump denote these times for A-Eye and the third umpire respectively; column % Red. represents how much percent A-Eye is faster than the third umpire.

have proved that A-Eye is almost as accurate as a third umpire, and in some cases, equally accurate.

On a side note, one might expect the accuracy of third umpires to be 100%, as they have ample time for decision-making. In this context, we believe that when A-Eye is applied to professional matches, it can greatly reduce this fallibility of the third umpires due to the lack of the human error.

### B. Run-Out Time

In this section, we compare the run-out times of AEye and 3Ump. To this end, we calculated the average run-out time for each sample, for both AEye and 3Ump. Our results are shown in Table III-B. Here, columns T_AEye and T_3Ump denote average run-out times for AEye and 3Ump respectively, and column % Red. shows the percentage reduction in this time brought about by AEye (as compared to 3Ump). We can see that, for each sample, A-Eye is extremely efficient as compared to 3Ump, and brings a reduction of approximately 100%. This result has a major contribution in validating our technology, and addresses the limitations discussed in Section I-A.

### IV. Conclusions and Future Work

In this paper, we have presented a novel technology, A-Eye, for the game of cricket, that is able to decide autonomously whether a batsman is out (or not-out) in a run-out situation. Typically, such situations require a decision by an off-the-field umpire called the third umpire. We have shown that A-Eye is extremely efficient as compared to the third umpire, and also, it is almost as accurate. It also has the potential to minimize the element of human error in the decision of the third umpires. Finally, it can estimate a rating for the performance of the field umpires in run-out situations.

*These results have convinced our local cricket council to employ our technology in a cricketing tournament to be held this year (2011).* Then, we will test A-Eye with diverse type of run-out scenarios that occur in professional cricket matches. Based on our own results, we believe that A-Eye’s accuracy will be further increased when it is employed with state-of-the-art camera technology to be used in this tournament. We also aim to calculate a rating for the field umpires of this tournament, through the Umpire Rater module. Once we have these results, we plan to send them to the ICC officials, in order to convince them to apply our technology at an international level.

Finally, the only limitation of our technology is that it cannot operate if the camera (for filming the run-out) is installed at a certain height from the ground. Hence, it requires a camera that is installed on the ground (at zero height). We don’t consider this as a limitation, because the introduction of a new technology always requires some sort of a change, e.g., the Hawk-Eye feature used in cricket requires the installation of four video cameras located at pre-specified angles and locations around the field [2].

### References


