Computer Vision in Cue Sports

Ross Kelso - 160002045

27th April 2020

University of
St Andrews

Senior Honours Project
School of Computer Science
University of St. Andrews

Abstract

Pool is a popular cue sport played in bars throughout the country. Players often need to recreate the table state in order to discuss strategy and shot selection with each other. This project aims to allow players to easily capture the state of the game by photographing a table (without restriction of the camera angle). This capture can then be used to recreate the table state in a 3D model allowing players to discuss various possible strategies from that position. Additionally, the output from this program could be used for a variety of other applications including virtual camera angles when streaming pool matches or the automatic refereeing of pool games.

Declaration

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated.

The main text of this project report is 9,110 words long, including project specification and plan.

Contents

1 Introduction

The aim of this project is to use computer vision to determine the position of pool balls on a pool table (as if viewed from directly above). This will involve detecting the balls on the table and the bounds of the table in a given image and then undoing the effects of perspective to determine the positions of the balls relative to the width and length of the table. Additionally it will be necessary to classify each detected ball by colour.

In this project I will be primarily looking at British black ball pool the version of pool which is commonly found in bars in the UK. A glossary of pool terminology is provided in appendix D. This terminology will be used throughout this report. When 'any angle' is stated in this report it means that the angle the photo is taken from is not respected to a view (such as top down) but the camera must still be looking down on the table such that the surface of the table can be seen.

Once the positions of the balls have been determined this data can then be used for a variety of applications, such as input to another program that can create a 3D render of the pool table that can be viewed from any angle to aid players with shot selection and strategy. This information could also be used to determine whether the a player is currently in a total snooker. Once this has been achieved for pool this technology could be applied to other cue sports such as snooker. The primary aims of this project are:

- To detect the bounds of the table in the input image
- To detect the position of the balls in the input image
- To classify the detected balls by colour
- To calculate from the above information the position of each ball on the table

The secondary aims of the project are:

- To correctly detect and differentiate balls that are occluding each other or the cushion
- To use the ball positions to create a 3D model of the pool table that can be rendered from any angle
- To programmatically referee a simple cue sport

Finally the tertiary aims are:

- To be able to determine what happened during a shot (e.g. if a ball hit a cushion, which object ball was hit first etc.)
- To be able to determine whether the detected balls are touching
- To automatically referee a pool or snooker match

The input to this program is an image of a pool table (from any angle) and the output is the position of all the balls on that table. There are 2 main stages to this project: detection and transformation. The detection stage is where the input image is processed to determine the location of the bounds of the pool table and the positions of the balls on the table. The transformation phase is when the perspective of the image is undone giving the positions of the balls relative to the table. Additional stages may be added where this information is used as input to another process in order to analyze or render the pool table.

1.1 Coronavirus

Due to the coronavirus pandemic I was unable to capture additional data that could have been used to evaluate and test this project. I had planned to capture more 'unseen' input data in order to evaluate the program against data that had not been used to design it. Additionally, I intended to capture data specifically for evaluating the accuracy of the ball detection (by more accurately measuring ball positions in the input image) and snooker detection by capturing data related to edge case snookers.

Instead of using this data I have used what relevant data I already had to evaluate the performance of these systems. Due to this the exact error in the program could not be determined as the exact positions of balls in the input image is unknown. I also used available data to test the snooker detecting program, however no borderline snookers exist in the existing data set so I was only able to test this for a trivial snooker.

2 Requirements Specification

The requirements of this project are to create a program that takes a photo of a pool table as input and gives the positions of all the balls on the table as output. This output can then be used by other programs to render the pool table from any angle or programmatically analyse the game. An additional requirement of this project is that the input image can be taken from any camera angle (where the table is the primary focus of the image).

In addition to these functional requirements there are also several non-functional requirements of this project. This project must be able to determine the ball positions with a high degree of accuracy as some applications will require this in order to be useful. For example in some cases a player may think they are totally snookered but they actually have the ability to hit an extremely small section of the object ball meaning they are not actually totally snookered, in this scenario there is a very small margin for error. Therefore, for a referee program to be useful the detected ball positions must be extremely accurate.

Furthermore the program ideally would be able to process a single image very quickly. This is so that it is possible for the program to keep up with real time video which will be necessary for some applications. In the game killer, for example, it is sufficient to look at the ball positions only between shots, but in pool and snooker it is necessary to watch the whole shot and determine what happened while the balls were moving. Not all steps of the program must meet this criteria, if the camera is static then the table bounds could be found once and only the ball detection and perspective transform need to happen each video frame.

In addition to the previous requirements in order to make this project as universally applicable as possible no special markers or setup from the user of any kind should be required.

3 Previous Work

In this section I will discuss previous works in the area of using computer vision to detect and track pool balls on pool tables. Previous work has predominantly been done on American pool tables. American pool tables tend to be blue and they use numbered 'spots and stripes' balls as opposed to British pool where the table tends to be green and the balls are reds and yellows.

3.1 Pool-Aid

Pool-Aid is a computer vision project that uses a top down view of a pool table in order to detect the ball positions and suggest potential shot choices to the user [1]. Pool-Aid requires the camera to be directly above the table with the entire table in frame.

Pool-Aid suggests different methods for detecting the bound of the table in their paper. The first step is to remove all sections of the image that are not the colour of the table by the threshholding the table based on colour. This creates a binary image where the pixels in the 'table colour' range are true and the rest are false. Contour detection is used to find the largest contiguous 'table coloured' area of the image. Once this contour is found the minEnclosingRectangle function in OpenCV is used to create a rectangular bounding area. They suggest that a possible future algorithm would use a linear Hough transform and cluster the detected lines would be a better solution. This solution is explored in this project.

In order to detect the position of the balls on the table Pool-Aid uses a background subtraction and contour finding

technique. Pool-Aid has a calibration period where the program averages the surface of the pool table to create a reference background image. During use this reference background is subtracted from the input image which removes the table surface from the image, leaving just the pool balls. The resulting image is thresholded which creates a binary image where the true values are the areas that did not contain the reference background, which should be where the balls are. Contour detection is run on the thresholded image, contour detection finds the boundary lines around the border of contiguous areas of the thresholded image. These contours are then filtered based on area and circularity to remove false positives. The contours that meet these criteria can then be considered to be the pool balls. The main advantage to this kind of contour detection is that the detection is not based on finding circles in the image, this means that the technique will still work when the balls are moving and the motion blur causes them to appear non-circular.

This work also had the added aim of recommending shot choices to the user but this is not relevant to this projects aims.

3.2 Automatic Pool Trainer

The Automatic Pool Trainer (ATP) is an automated system that is designed to help users learn how to play pool[4]. This system automatically detects the positions of the balls on the table in order to provide training exercises. This system uses a top down camera and calibrates its rotation and translation with the diamond markings which are found around the edge of American pool tables. In order to detect the ball positions a reference image of the table is subtracted from the input image to isolate the image areas containing balls. This image is then thresholded and contour detection is used to determine the location of the balls on the table as in Pool-Aid. Automatic Pool Trainer also provides training information for pool players, which is not relevant to the aims of this project. Overall ATP uses a very similar process to Pool-Aid.

3.3 Go Cardless

In a Go Cardless hackathon a team produced a pool ball tracking program intended to be used to keep score of pool games[5]. This work also uses the colour thresholding and contour detection techniques of the Automatic Pool Trainer and Pool-Aid in order to find the locations of the balls. This work also uses a top down angle of the table.

4 Software Engineering Process

For this project I used an iterative development method. This project is divided up into a few key steps: bounds detection, ball detection and transformation. Initially simple versions of each of these sections were developed which were then iterated and improved on by adjusting parameters or using more complex algorithms. Because each stage is independent from one another they can each independently be improved without effecting the others. Simplified versions of each step could still be used together to produce a prototype but iterating on each step individually increased the accuracy of the final output.

This method was used because it allows focus to be moved to the areas which are contributing the most error while still keeping the remainder of the program functional. Additionally, it works well for an open ended project as a fully functional program exists at each stage allowing iteration as much as possible in the time allowed whilst never having a non-functional submission.

An example of this iterative process is the various bound detection algorithms that were used throughout. Originally the contour finding methods of previous works such as Pool-Aid were used to find the bounds. This was an extremely inaccurate technique but allowed a working prototype to be created early on. Later iterations used these 'approximate bounds' as part of the pre-processing for more accurate bounds finding algorithms.

5 Ethics

The only ethical considerations for this project were getting proper consent from any participants who were filmed or photographed while capturing the input data for this project. The full ethics application and approval letter are included in appendix E.

6 Design

The main difference between this project and the previous works described above is that the aim of this project is to be able to detect the position of the pool balls on the table where the photo can be taken from any angle (i.e. is not restricted to a top down view). This has the added challenges that balls will be occluded by the cushions and each other, as well as the fact that perspective effects the positions that the balls appear in the image. In order to determine the position of the balls on the table this perspective transform will have to be undone.

Undoing the perspective will be done by creating a homography between the plane that is the surface of the table and the plane of the input image. Any 2 images of the same plane (in this case the surface of the table) are related by a homography that transforms the positions on one plane to the other. In order to undo the perspective created by the camera angle the homography between the table surface as it appears in the input image and the same plane as it would appear from a top down angle is determined. The findHomography function in OpenCV can be used to find the homography between 2 planes by using points of known position on both planes.

Therefore, in order to compute the homography several points of known position in the target perspective (top down) must be found in the input image. If the bounds of the table can be detected in the input image then these can be used to create the homography as their position is known in the output as they are the extremes of the dimensions of the table in the target plane. So that the output of this process is not bound to a particular table (with specific dimensions) the size of the table in the output is set to be 1 unit by 1 unit. Therefore the overall output will give the position of the balls as a proportion of the table width and table length.

6.1 Bounds Detection

The problem with using the actual surface of the table as the input plane for the homography is that the bounds of the slate are hidden under the cushions and therefore cannot be detected. An alternative is to detect the transition between the cushion and the wooden edge of the table. This transition is composed of several straight lines that can be detected using a Hough transform or RANSAC. Then to find the corners it is necessary to find the intersections between the detected lines.

Figure 1: Detected bounds

In figure 1 the blue line shows the detected bounds in one of the testing input images.

Because the detected bounds of the table will be used to create a homography that will be used to undo the perspective this means that the detected bounds should be as accurate as is possible as any error will cause the homography to be wrong. As this homography will be used to transform the position of each detected ball any error in the homography will be carried over into all the ball positions.

In order to detect the table bounds a similar approach to Pool-Aid and others is taken. First the image is converted into the Hue, Saturation and Value (HSV) colour space. All shades of green exist in a single section of the hue channel of this colour space. Therefore, a threshold can be applied to the hue channel creating a binary image where only the green sections of the image are true. Contour detection can then be used to create an approximate bounds' around the table by finding the boundary around the biggest contiguous area of the thresholded image. This essentially finds 'the biggest green thing' and assumes that that is the table.

In order to more accurately detect the bounds of the table (and therefore increase the accuracy of the result) edge detection is run on the thresholded image in order to find the pixels which are edges between the green table surface and the table boundary. These edge points are then used as the input to either the Hough transform or RANSAC algorithms which find straight lines in the image. The intersections between these lines can then be found to find the bounds of the table.

6.1.1 Hough Transforms

A Hough transform can be performed for any shape that can be described by a set of parameters. In the case of a straight line those parameters would be the gradient and intercept of the line, in the case of a circle the parameters would be the x position, y position and radius of the circle. The input to the Hough transform is a binary image where the edge pixels are true and the rest are false.

An accumulator matrix is created where each dimension is one of the parameters. The accumulator is quantised to some specified resolution (e.g if the intercept resolution was 1 then any line with intercepts within 1 pixel would be considered to have the same intercept).

Each edge pixel in the input image contributes to the accumulator values that could produce an edge in that location.

For example if the pixel (1, 1) is an edge that could be because a line with gradient 1 and intercept 0 is going through it, or because a line with gradient 0 and intercept 1 is going through it. Each possible of these lines is considered and the corresponding accumulator position in incremented. Once each edge pixel has been considered a threshold value is applied to the accumulator. Any accumulator position which meets the threshold is considered to be a detection.

Although this has been explained using the standard equation for a line:

$$y = mx + b \tag{1}$$

In practice line detection in Hough transform uses this equation for a straight line [6]:

$$\rho = x\ cos(\theta) + y\ sin(\theta) \tag{2}$$

$$y = \frac{\rho - x\ cos(\theta)}{sin(\theta)} \tag{3}$$

This is because in equation 1 the gradient is an unbounded parameter meaning the accumulator would have to be unbounded. Additionally, vertical lines can't be represented. In equation 2, θ is the angle between the x axis and the line and ρ is the distance from the origin. Both of these parameters are bounded, allowing the size of the accumulator to be bounded.

[3]

6.1.2 Random Sample Consensus (RANSAC)

RANSAC is an algorithm used to fit models (in this cases straight lines) to a set of data points. It is designed to be resilient to outliers.

RANSAC works by taking a random sample from the data points and using that sample to create a hypothesis of what the model could be. Other points in the data set are compared to this hypothesis using some loss function. If the loss for a given point is lower than some threshold then that point is considered an 'inlier'. If a given hypothesis has a lower total loss (for the inliers for that hypothesis) than the previous best hypothesis then it is saved as the best hypothesis. Additionally a minimum number of inliers is set to prevent a poor hypothesis from being considered. After a pre-defined number of iterations the best model (i.e. the line minimising the loss function with respect to its own inliers) is returned.

Because outliers are not counted towards the loss of a given hypothesis this makes RANSAC resilient to outliers. The outliers to contribute to how the models are fitted.

In order to detect the 4 lines that make up the edge of the pool table RANSAC will be run 4 times. Each time the inliers for the selected line will be discarded so they are not used when finding the next line. This will allow 4 distinct lines to be detected rather than the same line 4 times.

[2]

6.2 Ball Detection

In order to detect the balls on the table a circular Hough transform is used. It was chosen to use a Hough transform over the contour detection used in previous works because tracking moving balls is not necessary for the primary and secondary aims of this project and Hough transforms have the advantage that the entire object doesn't have to be visible in order to be detected.

Hough transformations can be done for a variety of different possible shapes. Where a shape can be described in terms of a set number of parameters a Hough transform can be done for that shape, where the Hough accumulator matrix has the number of dimensions as parameters. For example in the case of a straight line the parameters may be the gradient and intersection of a line. In order to detect the balls on the pool table we can use a Hough circle transform to find any

circles in the image by having the parameters of the Hough transform be the x and y coordinates of the center of the circle as well as the radius.

However, not all circles in the image will necessarily be pool balls since the pockets are round and there may be circular objects in the background of the photo. These types of false positive can easily be discarded by ignoring any circles detected outside the detected bounds of the table.

In a Hough transform all points around the edge of the circle will contribute to the accumulator for the detection of that ball. However, the accumulator threshold can be set such that a ball will still be detected even if only half of those points actually did contribute. This has the advantage that the Hough transform can detect balls that are partially occluded by each other or the cushion as long as the visible part of the ball is sufficient to reach the accumulator threshold.

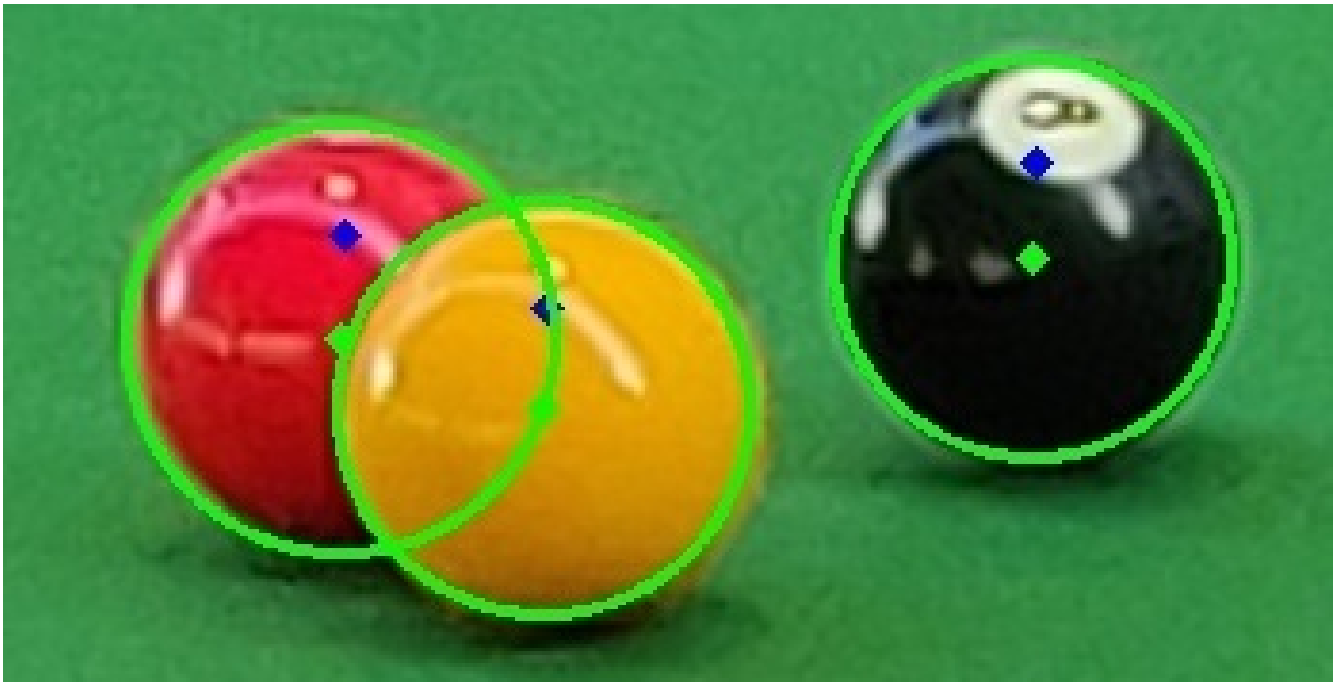

Figure 2: Hough transform 'seeing through' the yellow ball

When a ball is detected in this way by the Hough transform the ball is still being detected as a circle with position and radius because the curvature of the visible part of the ball is consistent with only 1 position and radius of the ball will still be detected in the correct place with the correct radius. Even if the centre of the ball is occluded. In this way the Hough transform is sort of able to 'see through' the cushion or other ball and accurately complete the balls position. This effect can be seen in figure 2.

It is necessary to set the accumulator threshold to a value that allows balls that are occluded to be detected but still discards false positives caused by noise. As a pool table is a very flat and plainly coloured surface the incidents of false positives on the table will be low. This allows the threshold value to be set low allowing balls to be partially occluded and still be detected.

The contour detection method used in other works does not have this advantage and so balls would not be properly detected when occluded.

6.3 Perspective Transformation

It is possible to use the detected bounds of the table to set up a homography between the table surface and a top down view of the table. However, the detected bounds positions are not on the flat slate surface of the table. The bounds are detected on a different plane that is level with the top of the cushions. When a ball is detected the center of the

detected circle is not the point where the ball intersects this plane (unless the image was taken from a top down view). It is therefore necessary to compute from the available information the point at which the detected ball intersects the cushion plane would appear in the input image.

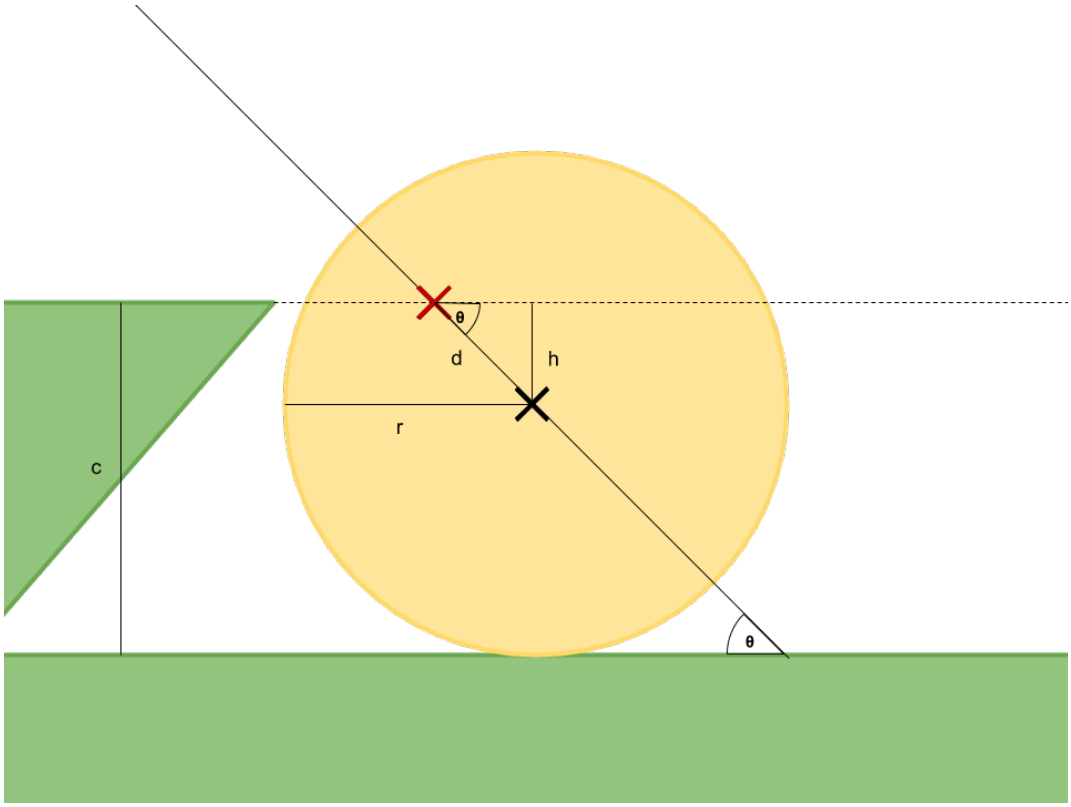

Figure 3: Accounting for cushion plane

In figure 3 the solid line represents the plane the image forms, the black 'x' represents the detected center of the ball in the image and the red 'x' represents the point where the ball intersects the cushion plane would appear in the input image. θ is the angle between the image plane and the surface of the table. h is the difference between the height of the cushion (c) and the radius of the ball (r). The target position is the position of the ball in the image plus the difference between the input location and the offset caused by the plane intersection (d).

d can be computed using the following equation:

$$d = \frac{h}{cos(\theta)} \tag{4}$$

However, d is measured in pixels (in the input image) where as the height of the ball and cushion is measured in cm. In order to compute d in pixels the value of h is converted from cm to pixels. In order to do this the ratio between detected diameter of the ball and the known physical diameter the ball is used as a conversion factor.

Additionally, if the camera was not held level when the input image was taken this should also be accounted for. In order to account for this the previously calculated offset (d) must be rotated to match. To do this consider the vector $\binom{d}{0}$, rotate this vector by the angle the camera was held at and add that to the detected center of the ball. This will produce the position the ball intersects the cushion plane in the input image as required.

In order to determine the values for these angles the OpenCV `calibrateCamera(..)` function can be used [6]. One of the outputs of this function is the desired angles.

Once the point where each ball intersects the cushion plane is found a homography can be used to transform the ball positions as found in the image into their positions on the table. This is the desired output of the program.

7 Implementation

For many of the computer vision algorithms used in this project the OpenCV implementation was used [6].

7.1 Pre-processing

The first pre-processing step is to convert the image from and RGB colour space into a Hue Saturation and Value colour space. This allows future operations to be performed on the hue, saturation and value channels separately. These channels have more semantic meaning than the red, green and blue channels. For example across the surface of a ball the value channel is a fairly consistent value regardless of the colour of the ball, however the red green and blue values would have different properties depending on the colour of the ball.

In order to filter out any false positives in any later steps a mask of the approximate bounds of the table is created. A double threshold is run over the hue channel of the image to binarize the image into 'green' and 'not green' pixels (call this the threshold image). Then contour detection is run on the threshold image and the contour with the largest area of its convex hull is taken to be the approximate bounds of the table. The convex hull of a contour is the area which contains the contour without being convex or going back in on itself. The convex hull is used rather than the original contour as the table is broadly rectangular and a rectangle is a convex shape, therefore the expected contour should be convex and any part that isn't is due to error. Intuitively this can be thought of as finding the largest 'green thing' which will be the table in any image where the table takes up a significant portion of the frame. This contour is then used to create a new mask for the approximate bounds for a table. This mask can be applied to the original image to create a version of the image with the background removed.

The approximate bounds contour is also used for automated calibration. Some of the parameters to the various computer vision algorithms are dependent on the size of objects in the image (as measured in pixels) this means that the resolution of the image and the size of the table in the frame of the image affect the required values for these parameters. To fix this problem some of these parameters are set by multiplying a constant value by the perimeter of the approximate bounds. This allows these parameters to scale with the size of the table in the frame making the program far more resilient to changes in camera and camera angle.

OpenCV provides an implementation of Canny edge detection [6]. Canny is run over the threshold image in order to find the edges of any 'green things' this should find the edges of the cushions where they meet the table (as well as many other false positives). In order to reduce the number of false positives the approximate bounds mask is applied to remove any edges in the backgrounds. The approximate bounds contour is shrunk by a factor of 10% and used to produce a second 'internal' mask, this removes edge detections caused by the edges between the balls on the table and the table surface. The edges image now only contains edges detected near the boundary of the approximate bounds. This edges image is used as the input to the bounds detection algorithm being used.

7.2 Bounds Detection

The bounds detection algorithms take the detected edge points found in the pre-processing step and output 4 points that make up the corners of the table. The first step of this is finding the straight lines that make up the edges of the table. The second step is finding the intersections between those lines in order to determine where the corners of the tables are. For the intersection step to work properly exactly 4 lines should be detected in the detection step.

7.2.1 Linear Hough Transform

The first method used to detect the straight lines in the image was a linear Hough transform. This algorithm works by using an accumulator where the dimensions of the accumulator matrix are the parameters of the model (in this case a straight line). Each point in the input image that is true contributes to all the possible places in the accumulator which would cause a line to go through that point. A threshold is used to determine which accumulator entries are considered a detected line.

This method is able to find the lines that make up the bounds of the table, the issue is that not exactly 4 lines will be detected. The solution to this issue was to set the threshold lower than it needs to be (this can cause some bounds to be detected multiple times with slightly different parameters) and then use another mechanism to reduce down the lines into exactly 4.

Initially a function called `simplify_lines(..)` was written that would consider how similar pairs of lines where and discard one of the lines that was most similar to another until there were 4. However, this is not necessarily the 'best' detection of the given table edge. Later this was improved by using K-Means to cluster the lines into 4 categories and take the means of the clusters found to be the 4 detected lines [7].

Another solution would be to use the accumulator value for each line and take the 4 lines with the highest accumulator values. However it is not necessarily true that the 4 lines with the highest accumulator values are the 4 different edges of the table. The longer edges of the table contribute to the accumulator more because there are more points along them. Therefore the 4 highest accumulator values may be multiple detections of the longer edges and not include the short edges.

None of these approaches is ideal but K-means is more resilient as the output lines are an aggregate of all the detected lines. This algorithm is used when running in Hough lines mode, which is the default.

OpenCV computes the lines in the form of equation 2 [6], the lines are converted into the form of equation 1.

7.2.2 RANSAC

RANSAC was implemented as a python module based on the psuedo-code in the RANSAC Wikipedia article [2]. In order to run RANSAC the edges image is converted into a list of points, this takes approximately 3s to complete and is therefore a significant contributor to the additional processing time of using RANSAC. Each pixel in the edge image is added to a list which is a high complexity operation.

Additionally, RANSAC is run 4 times to detect the 4 lines making up the table bounds. Each time a line is detected the inliers for that model are removed from the list of points which the next RANSAC iteration is run on. This means that a lot of operations are spent modifying the list of edge points which could potentially be improved by optimising the data structures used.

This RANSAC implementation uses the distance between the point and the hypothesis as its loss function. Any edge point within 10 pixels of the closest point on the hypothesis line is considered an inlier. This loss function was used because it was straight forward to implement and gives a very good indication of whether a point is part of the line.

Additionally, an early stopping procedure was implemented in an attempt to reduce the run time needed when using RANSAC. This worked by setting an early stopping point where if a certain number of inliers were met the hypothesis was considered 'good enough' and would immediately stop iterating. This is not used in the final submission because it caused the quality of detected lines when using RANSAC to be much lower.

7.2.3 Intersections

In the previous step straight lines were detected in the form $y = mx + c$, to finish finding the table bounds these lines must be converted into the 4 corners points of the table. To do this the intersections between the lines are found by solving the equations for each pair of lines simultaneously. This results in more than 4 intersections as the intersections between lines that aren't adjacent on the table are also computed. This is necessary as it is unknown which line is which at this point. Any intersection that is not near the approximate bound contour is then discarded resulting in 4 points of intersection which are the corners of the table. Finally, the `order_points(..)` function is used to put these points into a standard order to form the bounds.

7.3 Ball Detection

There are 2 steps to ball detection: circle detection and colour classification. A Hough transform is used for circle detection and a simple colour classifier has been implemented for this project.

7.3.1 Circle Detection

Circle detection uses the OpenCV implementation of a Hough circle transform. The input image is the value channel of the HSV version of the input image. The edge detection is not specifically run because the OpenCV implementation of Hough circle detection has its own edge detection built in [6]. The minimum and maximum ball radii is set as a proportion of the perimeter of the approximate bounds to be resilient to different cameras and the size of the table in the image frame. The minimum distance between detected circles is set to a proportion of the minimum radius. The minimum distance prevents the same ball from being detected multiple times.

To reduce the number of false positives the input image is blurred and masked (using the approximate bounds mask) before being used. The mask removes any false positive circles that may be detected in the background and the background. Blurring the image reduces the chance of noise in the image causing any false positives.

7.3.2 Ball Classification

A simple ball classification algorithm was implemented to determine the colour of detected balls. The colour of the ball in the RGB image is averaged to produce a single RGB value. Each of the red, green and blue channels is then divided into whether the value is high or low, based on whether the value is higher or lower than the mid-point 128. If all 3 channels are high the ball is considered white (i.e. is the cue ball), if only red is high then the ball is red, if only red and green are high the ball is yellow, if all channels are low then the ball is black. In any other case the ball is considered unknown colour. This is a very simple classifier but it works very effectively. The colours of the balls used in pool are very spaced out in the colour spectrum allowing a technique like this to be used. This also makes the classifier very accurate as there is a very large margin between the different colours.

However, the drawback to this classifier is that it doesn't generalise well to other cue sports. In snooker there are more coloured balls which are miss classified by this algorithm. For some of the colours used in snooker a similar approach may work but not in all cases. It is likely that the pink and brown may be confused for reds, or the brown for the yellow depending on the algorithm. In a game like snooker with more colours to detect a better approach may be to average the hue of the detected ball and then split the hue spectrum up into classification regions. However this approach may need special cases for the black, white (as these can be any hue) and pink (as pink is a light shade of red so may have a similar hue to the red balls).

Classifying American pool balls (spots and stripes) would be much harder. There are 2 balls of each colour (1 spot and 1 stripe) so the classifier would have to be able to distinguish between more colours and balls of the same colour where one is striped and one is solid. Pool-Aid created an American pool ball tracker as part of their project [1]. A similar technique to theirs could be used if this project was extended to American pool. The Automatic Pool Trainer avoided this problem by using a yellow object ball despite being played on an American table [4].

Figure 4: Resulting bound and ball detections

7.4 Perspective Transform

In order to transform the position of the balls a homography is set up between the detected bounds of the table and an imaginary plane where the table is a unit square. Therefore, when the balls are transformed by this homography the output is the position of the ball as a proportion of the table width and table height.

In order to transform the balls the point the ball intersect the cushion plane must be calculated as described in section 6.3. Any conversion between pixels and cm is done by using the ratio between the detected ball radius and the physical ball radius. The other required parameter is the angle between the table and the camera. In order to obtain this angle the camera is calibrated using the OpenCV calibrate camera function [6]. Normally this uses a chess board to calibrate the camera but ideally a separate camera calibration step would not be necessary so the detected table bounds are used to calibrate the camera. Calibrating the camera returns the angles used in the calculation as described in section 6.3.

Once this has been accounted for the balls position is transformed using the homography to determine its position on the table and the primary aims of this project are complete. A number of outputs are generated including a top down view of the table and the positions of the balls as represented in JSON (which can then be used as the input to another program).

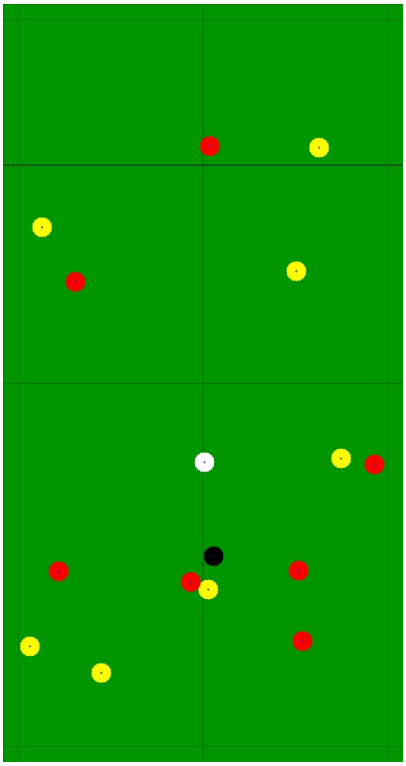

Figure 5: Resulting Transformation

7.5 3D Rendering

In order to create a 3D render of the pool table the library vpython was used[8]. A program was written that takes the output of the detection program and uses it to create 3D sphere objects in the right place on a 3D table. The user can then interact with the 3D render and view the model from any angle.

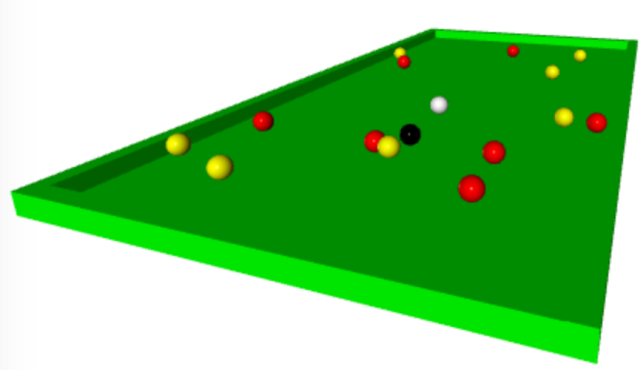

Figure 6: 3D Render

7.6 Snooker Detector

The second program that has been written to programmatically analyse the ball positions is the snooker detector. This program takes the ball positions as input and tests whether the current player is snookered. It does this by considering possible paths between the cue ball and the target ball and testing if these paths are obstructed. The paths it checks are the 2 most extreme cuts on each side of the target balls and hitting the target ball directly in the middle. By testing

in this way it is possible to determine whether it is in any way possible to hit the target ball from the current cue ball position. If it is not possible to hit any of the target balls then the player is considered snookered.

Ideally to test and evaluate the performance of this snooker detector additional data would have captured where the balls are specifically set up in edge case and interesting snookers. Unfortunately this is not possible because of the ongoing coronavirus pandemic. However, if the 'break' image the current player is not snookered in any colour and in the 'setup' image the player would be snookered on black (although the player would not be on black the snooker condition can still be tested). These images were used to test the snooker program.

8 Evaluation and Critical Appraisal

To evaluate the error of this project the known position of the black ball in the 'setup.jpg' and 'straight-on.jpg' images was used and compared to the detected position of the black ball.

The black spot on the pool table is in the middle on the x axis and $\frac{3}{4}$ of the the table length on the y axis. Therefore the expected position of the black spot is 0.5 table widths and 0.75 table lengths. When the program is run on 'setup.jpg' and 'straight-on.jpg' where the black ball is sitting on the black spot and so the distance between the detected position in this ball and the expected position of the black spot is the error in the detected position. The program outputs the distance between the detected black ball and the black spot in order to measure this error.

As RANSAC is a stochastic algorithm the program was run 5 times to make the results more reliable. Values have been rounded to 3 decimal places.

Input	**Algorithm**	**Error x (cm)**	**Error y (cm)**	**Magnitude (cm)**
Setup	Hough	0.448	-0.374	0.584
	RANSAC	0.088	-0.683	0.689
		0.079	-0.728	0.732
		0.128	-0.675	0.687
		0.119	-0.644	0.655
		0.150	-0.816	0.830
	RANSAC (avg)	0.113	-0.709	0.719
Straight On	Hough	0.206	-0.127	0.242
	RANSAC	0.251	-0.577	0.630
		0.248	-0.463	0.525
		0.272	-0.423	0.503
		0.212	-0.457	0.504
		0.247	-0.442	0.506
	RANSAC (avg)	0.246	-0.472	0.534

Unfortunately it is impossible to know the true amount of error produced as the black spot on this table is not necessarily in exactly the right spot and when the black ball was placed it may not have been placed in exactly on the black spot. Therefore there is an unknown error in the black balls 'known' position. I had intended to measure the exact position of the black spot and capture new data in order to more accurately measure the error of this program however this is impossible due to the ongoing coronavirus pandemic.

In order to visualise the direction of the error these graphs were created:

Figure 7: Black ball position error

From the above data it would appear that the Hough transform method of bounds detection is more accurate, however it is impossible to make this conclusion as the true position of the black ball might be between -0.5cm to -0.7cm away in the y direction away from where it should be which would mean RANSAC is more accurate. It appears that the error is always in the same direction which would support this hypothesis.

In these input images other balls on the table have also been deliberately placed on the baulk line and exactly in the middle of the table. These have not been used to measure the error in this program because the baulk line is visibly not straight in person and there is no middle spot. Therefore, the positions of the other balls cannot be known with any degree of certainty.

The error reported when using RANSAC varies. This indicates that from run to run the stochastic nature of the algorithm introduces some error. It may be possible to reduce this by altering the parameters to RANSAC so that more points are required to be inliers and by running more iterations.

Additionally, as RANSAC is stocastic so in some instances it will not successfully detect valid bounds and the program will fail. The effect of this could be reduced by increasing the number of RANSAC iterations, however this would cause the program execution to take longer. For these reasons Hough transform is the default bounds finding algorithm.

However, RANSAC is more robust to different input images. For example the 'test.jpg' image is a hard case because the table makes up a small proportion of the image. In order to successfully detect bounds in this image RANSAC must be used. Balls that are close together are also not detected in this image likely because the minimum distance parameter to the Hough transform is too large. However, lowering it causes double detections of the other balls before the missing balls are detected.

Finally, in order to determine that the camera does not need to remain level I used photo editing software to rotate the 'setup.jpg' image to create the 'rotated-5.jpg' and 'rotated-neg5.jpg' images. These simulate what the photo would look like if the camera was rotated by 5 degrees in either direction. These demonstrate that the program can account for this properly as the error is not significantly different when run on these images.

9 Conclusions

This project is able to determine the location of pool balls on a table to reasonable level of accuracy however the precise error is unknown. This meets all of the primary aims. In addition, the program is resilient to changes in camera angle and can accurately detect balls even when occluded. Additionally, the output of this program can be used to render the pool table from any angle. Therefore, this project meets the first two secondary aims.

Additionally, the snooker detector program is a step towards automatic refereeing of cue sports, and therefore partially fulfills the tertiary aims (although much more work would be needed to complete these aims). As it stands this snooker detector is not massively useful because most snookers that would be in dispute by any players would be due to very narrow margins. As the exact accuracy of this project is unknown (and may be up to a centimeter) this snooker detector would not be able to resolve any dispute around a very borderline snooker as any conclusion could easily be swayed by the error. However if further testing and/or improvements demonstrated that the program was sufficiently accurate then this snooker detector could potentially be very useful for helping to referee the early rounds of tournaments where there are more matches than available referees.

The game Killer is a simple cue sport where each player has 3 lives. Players are called up in a random order and must pot a ball to retain that life if they miss they lose that life. As the only information needed in this game is whether a ball was potted this game could be refereed automatically if photos of the table were provided to the program between shots. Any ball that is no longer detected could be considered potted. This would however require a lot of human intervention. Ideally a more advanced version of this program would be able to automatically detect when the balls stop moving and automatically register the end of the shot for this purpose. However this does mean the project partially meets the final secondary aim.

In order to meet the tertiary aims the program would have to be extremely accurate (to determine if balls were touching and to call snookers). Additionally the program would have to be able to track the balls while they were moving in order to determine what occurred during the shot to determine whether or not that shot was legal.

The key achievement of this project is improving on previous cue sports detection systems like Pool-Aid. These systems are restricted to a top down camera angle which isn't always possible or convenient. However, the contour detection method used by Pool-Aid for detecting balls, while less accurate for occluded balls is able to track balls that are in motion. Future work in this area may be to combine the two techniques to allow balls to be detected when occluded but also when in motion. This would potentially create an extremely versatile and resilient system that could be used in a variety of applications including completely automated refereeing.

10 Acknowledgments

I would like to thank my supervisor Dr Ognjen Arandjelović. Additionally I would like to thank the University of St Andrews Students' Association for allowing me to capture photos and video of their pool tables as well as the members of the University of St Andrews Pool and Cue Sports Society who took part.

References

[1] Samuel Bauza et al. "POOL-AID: Detection and Identification of Pool Balls for the purpose of recommending shots to beginner players." In: (2016).

[2] Wikipedia Contributors. *Random sample consensus*. Last accessed 16th April 2020. 2020. URL: https://en.wikipedia.org/wiki/Random_sample_consensus.

[3] Peter E Hart and RO Duda. "Use of the Hough transformation to detect lines and curves in pictures". In: *Communications of the ACM* 15.1 (1972), pp. 11–15.

[4] Lars Bo Larsen et al. "Development of an automatic pool trainer". In: *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*. 2005, pp. 83–87.

[5] Harry Marr. *Hacking on Side Projects: The Pool Ball Tracker*. Last accessed 22nd April 2020. URL: https://gocardless.com/blog/hacking-on-side-projects-the-pool-ball-tracker/.

[6] OpenCV. *OpenCV 4.1.1 Documentation*. Last accessed 20th April 2020. 2019. URL: https://docs.opencv.org/4.2.0/.

[7] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[8] David Scherer and Bruce Sherwood. *VPython*. Last accessed 23rd April 2020. URL: https://www.vpython.org/.

Appendices

A Testing Summary

This program was tested by using the testing input images. Each stage was tested individually by adding additional UI windows showing intermediate image and pre-processing steps. This allowed me to adjust parameters and algorithms for each step individually.

Additionally, comparing the output images to the known positions of the balls on the table was used to determine if changes were making the system more or less accurate. In accurate detections, false positives and false negatives can easily be seen in the 'detections' output image.

B User Manual

This project is implemented using Python 3 and several libraries. Open a terminal in the root folder of this project (i.e. the one with src, input etc.). To install set up and activate a virtual environment:

```
virtualenv venv
source venv/bin/activate
```

Install the dependencies specified in the requirements.txt file using pip:

```
pip3 install -r requirements.txt
```

To run the program run:

```
python3 src/main.py input/break.jpg
```

The only required parameter of the program is the path to the input image. A number of input images are provided in the 'input' folder. If a directory is specified each file in that directory will be input into the program in turn.

By default the program will open a number of output windows showing the thresholded image, the edges image, ball and bounds detections ('detections') and a top down view of the table ('result'). The 'detections' image and 'result' images will be save to the output folder (by default 'output'). Additionally, a 'positions.json' file will be saved to the output folder.

To exit the program press any key with one of the output windows in focus. If you close all the output windows (using the 'x') the program will get stuck executing and will have to be killed. This is a quirk of how OpenCV image windows work that I don't know how to fix.

Optional parameters include:

- `-h` - this parameter will show help information
- `-r` - this parameter will rotate the output 90 degrees each time it is specified. e.g. `-r` rotates 90 degrees, `-rr` rotates 180 degrees and `-rrr` rotates 270 degrees. Use this option if the output image is not in the correct orientation.
- `-o` - this parameter allows you to change the output folder. Note: the output folder must already exist before the program is run, it will not be created

- `--render` - this parameter will create a 3D render of the pool table. The render will open in a browser window, close this window to exit the program. Hold down your right mouse button to rotate around the table, use your scroll wheel to zoom.
- `--ransac` - this parameter causes the program to use RANSAC to find the table bounds instead of the Hough transform which is used by default
- `--headless` - this will prevent the program from opening output windows. This does not effect the 3D render

Optionally the 3D renderer can be run separately:

```
python3 src/render.py output/positions.json
```

The 3D render has 1 required argument and that is the path to the 'positions.json' file to use as input.

The snooker detector program can be used to determine if the current player is snookered in a given table state. To use it run:

```
python3 src/snookered.py output/positions.json
```

The snooker detector has 1 required argument and that is the path to the 'positions.json' file to use as input.

All programs use 'config.py' as input for information about the dimensions of the pool table and size of the balls. This config is correct for the pool table in the images provided.

C Example Run

Figure 8: Input image

Figure 9: Detections

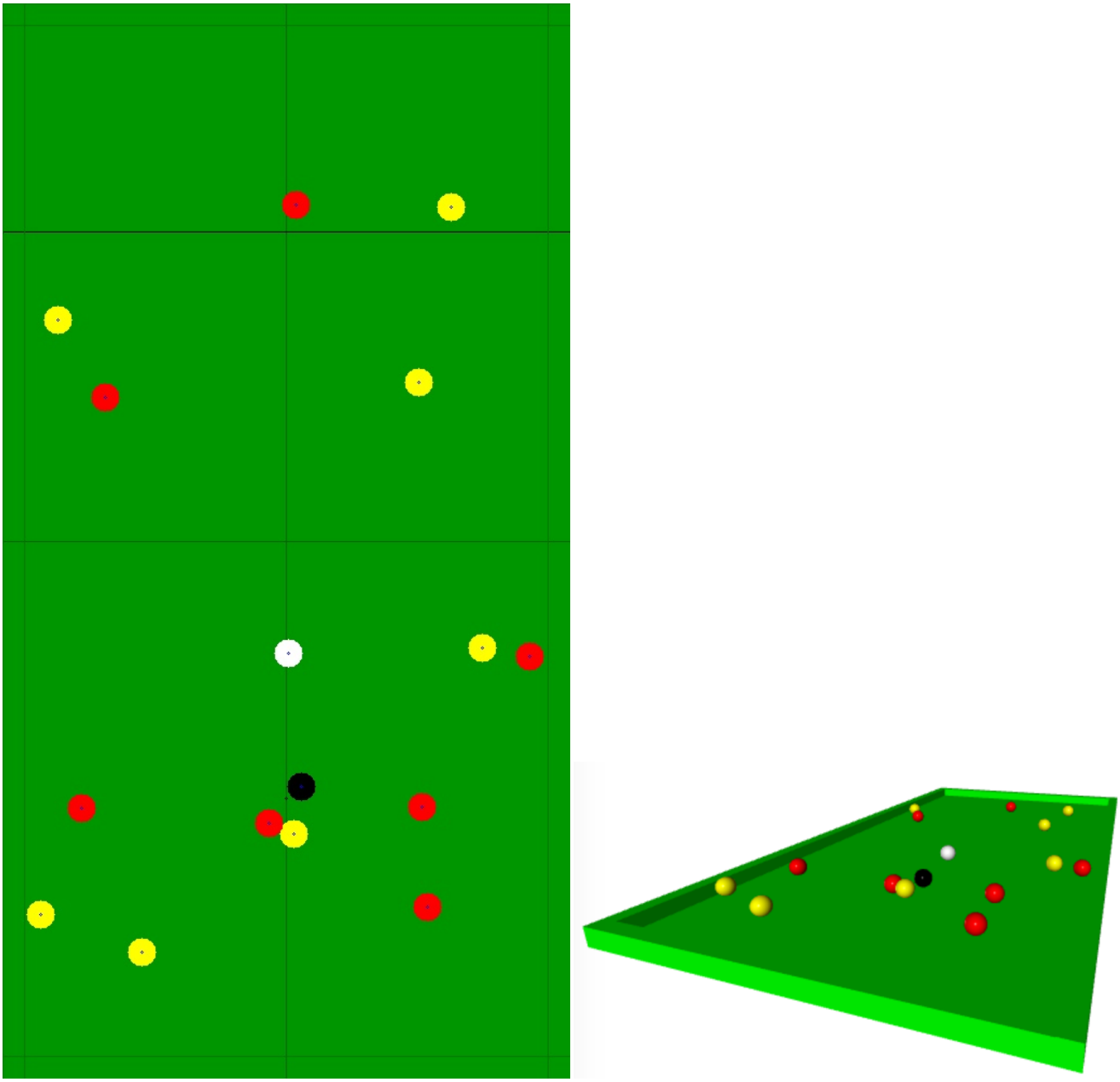

Figure 10: Results

D Pool Terminology

The game is composed of a table with 6 pockets, 7 red balls, 7 yellow balls, a black (8) ball and a white cue ball. At the beginning of the game each player is given a colour (either red or yellow), they are said to be 'on' that colour. The objective is to pot all the balls of your 'on' colour and then the black.

Break The 1st shot of a frame (or game) which breaks up the rack of balls

Cloth The green fabric covering the slate and cushions

Cue The cue is the stick you strike the cue ball with

Cue Ball The white ball, that you strike with your cue to play a shot

Cushion The cushions are boundary of the play area between pockets

Cushion Foul A foul caused by no ball hitting a cushion during a shot

Cut A cut is a shot which hits the target ball very close to the edge (rather than close to the middle)

Foul When a player breaks a rule or fails to play a legal shot and are penalised by awarding their opponent 2 visits (i.e. their opponent may play twice or gets "2 shots" that "carry")

Frame A game of pool

Killer The game Killer is a simple cue sport where each player has 3 lives. Players are called up in a random order and must pot a ball to retain that life if they miss they lose that life. The last player remaining wins.

Legal Shot For a shot to be legal the player must hit an 'on' object ball with the cue ball and then one of the balls must hit a cushion or be potted

Object Ball The red, yellow and black balls

Shot Striking the cue ball with the cue

Slate The flat playing surface of a pool table is a rectangular slate covered in cloth

Total Snooker When it is impossible to play the cue ball in a straight line and hit an 'on' ball

Visit After successfully potting an 'on' ball that player goes again. A visit is a continuous chain of potting 'on' balls without missing.

E Ethics