The University of Dublin

TRINITY COLLEGE

HockeyCam: An Android App that analyses and records Hockey footage in real-time

Stuart Rigby-Jones
B.A. (Mod.) Business and Computing
Final Year Project April 2013
Supervisor: Fergal Shevlin

School of Computer Science and Statistics

O'Reilly Institute, Trinity College, Dublin 2, Ireland
DECLARATION

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university

_______________________________  __________________________
Name                                      Date
**ACKNOWLEDGEMENTS**

I would firstly like to thank my project supervisor, Fergal Shevlin, for his help, guidance and direction throughout my project.

I would also like to thank Three Rock Rovers Hockey Club for the use of their facilities.

I would finally like to thank my classmates, friends and family for their support during this project.
ABSTRACT

The aim of this project is to design an Android application that will return video footage of a selected area of a hockey pitch in real time.

The Android device will be mounted on the goal frame, and will be analysing the hockey pitch in real time. Once there is sufficient activity within a certain area of the pitch, the footage will be recorded as a video file and available immediately for playback. This will allow users to instantly watch back and analyse action in the most important areas of the pitch.
# Table of Contents

ACKNOWLEDGEMENTS .................................................................................................................. 3

ABSTRACT ....................................................................................................................................... 4

1. INTRODUCTION .......................................................................................................................... 7

1.1 Motivation ..................................................................................................................................... 7

1.2 Project Approach and Design ....................................................................................................... 11

  1.2.1 Field Hockey: Rules and Explanations ................................................................................. 11

  1.2.2 Background of Project and Existing Technologies ................................................................. 12

  1.2.3 Use of Android Platform ...................................................................................................... 13

  1.2.4 Use of OpenCV .................................................................................................................... 13

  1.2.5 Design Camera Interface ...................................................................................................... 13

  1.2.6 Analysing Camera Frames .................................................................................................... 13

  1.2.7 Storing frames as video.......................................................................................................... 14

1.3 Overview of Project ...................................................................................................................... 14

  1.3.1 Libraries and APIs used ....................................................................................................... 14

  1.3.2 Application interface ............................................................................................................ 15

  1.3.3 Camera Frame Analysis ........................................................................................................ 15

  1.3.4 Convert the frame to a video file ......................................................................................... 16

  1.3.5 Storing the video file ........................................................................................................... 16

2. DESIGN AND IMPLEMENTATION ................................................................................................. 17

2.1 Systems, Languages and APIs used ............................................................................................ 17

  2.1.1 Issues with Libraries and APIs .......................................................................................... 17

  2.1.2 Working with Android ......................................................................................................... 19

2.2 Implementation of Project ........................................................................................................... 20

2.3 Camera Use and Interface .......................................................................................................... 21

  2.3.1 Using the Camera to record Video .................................................................................... 21
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3.2</td>
<td>Processing camera frames</td>
<td>22</td>
</tr>
<tr>
<td>2.4</td>
<td>Semi-Circle and Motion Detection</td>
<td>23</td>
</tr>
<tr>
<td>2.4.1</td>
<td>Finding the D in the image</td>
<td>23</td>
</tr>
<tr>
<td>2.4.1.1</td>
<td>Canny Edge Detection</td>
<td>24</td>
</tr>
<tr>
<td>2.4.1.2</td>
<td>Dilation and Erosion of Canny Image</td>
<td>25</td>
</tr>
<tr>
<td>2.4.1.3</td>
<td>Hough Lines</td>
<td>26</td>
</tr>
<tr>
<td>2.4.1.4</td>
<td>Detecting the D</td>
<td>29</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Comparing Camera Frames</td>
<td>30</td>
</tr>
<tr>
<td>2.5</td>
<td>Video Storage</td>
<td>31</td>
</tr>
<tr>
<td>2.5.1</td>
<td>Storing the video</td>
<td>31</td>
</tr>
<tr>
<td>2.5.2</td>
<td>Checking duration of videos</td>
<td>33</td>
</tr>
<tr>
<td>2.6</td>
<td>Positioning of Camera</td>
<td>34</td>
</tr>
<tr>
<td>3.1</td>
<td>Limits and Constraints of Project</td>
<td>36</td>
</tr>
<tr>
<td>3.2</td>
<td>Successes</td>
<td>37</td>
</tr>
<tr>
<td>3.3</td>
<td>Difficulties</td>
<td>38</td>
</tr>
<tr>
<td>3.4</td>
<td>Future Work</td>
<td>39</td>
</tr>
<tr>
<td>4.</td>
<td>Conclusion</td>
<td>41</td>
</tr>
<tr>
<td>APPENDIX</td>
<td></td>
<td>42</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td></td>
<td>43</td>
</tr>
</tbody>
</table>
1. INTRODUCTION

1.1 Motivation

The topic of sports analysis and technology within sport is one of great contention and debate in current society. Recent advancements in areas such as Computer Vision and Motion Sensing technology has meant that there is now a pressure on professional sports bodies to introduce relevant systems into their sports, with the view that this would eliminate the ambiguity that professional sport is occasionally won or lost by.

Indeed, one of the foremost technologies on the market at the moment is Hawk-Eye. Hawk-Eye was developed in England by Roke Manor Research Limited, and was launched independently in 2001 (Hawk-Eye, 2013). The technology uses high speed cameras placed strategically around a sports arena, and tracks the trajectory of the object in question – usually a ball. It then calculates the most likely path statistically that the ball will take based on its trajectory and velocity, and returns the ball’s position when it reaches a predetermined destination – for example, when a tennis ball comes into contact with the court. This technology has the ability to remove any ambiguity to a very high accuracy, if there is ever a case where the ruling officials are unable to make a definitive decision by eyesight alone.

Hawk-Eye is probably the most successful, prominent example of technology being used in sport at the moment. It has been successfully introduced into tennis and cricket, and is now an integral part of the adjudication process in both sports. Below are some examples of how Hawk-Eye is implemented in sport.
Fig 1. – “Hawk-Eye in Cricket 1” -
http://images.techtree.com/tpimages/story/68507_rw_bbc05_full_hawkeye_small.jpg

Fig 2 – “Hawk-Eye in Cricket 2” -
http://sportsmeter.files.wordpress.com/2012/01/lbw.jpg
Fig. 3 – “Hawk-Eye in Tennis 1” - [http://www.matthewreedcoaching.com/wp-content/uploads/2012/07/Hawk-Eye-in-tennis.jpeg](http://www.matthewreedcoaching.com/wp-content/uploads/2012/07/Hawk-Eye-in-tennis.jpeg)

Fig. 4 – “Hawk-Eye in Tennis 2” - [http://i.dailymail.co.uk/i/pix/2011/03/14/article-1366084-057B6F5F000005DC-462_468x286.jpg](http://i.dailymail.co.uk/i/pix/2011/03/14/article-1366084-057B6F5F000005DC-462_468x286.jpg)
Hawk-Eye is particularly interesting for this project as it is an example of the way that Computer Vision can aid professional sports. Computer Vision is used to track the ball and its whereabouts, which is then used to calculate its likely path. Systems like Hawk-Eye are becoming more and more commonplace in professional sport, and this was a major factor in the choice of my project.

For my project, I was working with the sport of field hockey, or simply hockey as it is referred to locally. Hockey is not a professional sport in Ireland, so therefore there are no widely implemented technologies in the game. This gave me a large amount of potential to work with in my project, and allowed me to approach the project openly and explore many different possibilities about how I would execute my project.
1.2 Project Approach and Design

In this section, I will provide an overview of the processes that I planned on using at the beginning of the project, as well as the main steps that would need to be taken during the project.

1.2.1 Field Hockey: Rules and Explanations

Field Hockey, known simply as hockey in Ireland, is an 11-a-side game, usually played on Astroturf pitches. Each player has a hockey stick which is used only to hit a hollow, solid ball (usually white), with the aim of getting this ball into the opposition goal. The pitch is designed as in the figure above. Each team usually lines up with a goalkeeper, 4 defenders, 3 midfielders and 3 forwards. Goalkeepers wear large amounts of padding so that they can safely stop the hockey ball from going into the goal. Balls can be raised off the ground if they are...
goal bound, or scooped into other areas of the pitch as long as it is safe for other players.

The pitch is divided into quarters of 25 yards each. Each semi-circle at the end of the pitch is known individually as the “D”. A goal can only be scored if the ball is touched last from within the D. This means that any notable action in the scoring of a goal will happen around the area of a “D”.

1.2.2 Background of Project and Existing Technologies

In Ireland, hockey is an amateur, non-professional sport. This means that there are few, if any, modern technologies currently being used. At a professional or international level, there is one system that is in the process of being implemented, but is still being tested and is not commonplace. This system uses a referral method, whereby an umpire or team captain can ask for a decision to be reviewed. An extra match official will then replay camera footage of the sequence of play in question, and return a verdict of whether the on-field decision should stand or if it should be overturned, much like the Television Match Official (TMO) system used in rugby union and league. The system in hockey was trialled in the London Olympics in 2012, while a slightly different system was used in the Beijing Olympics in 2008. The current system is also in the process of being tested in the European Hockey League (EHL), which is the premier club competition in Europe.

The system used in hockey uses purely cameras to facilitate the decision. It is therefore constrained by the number of cameras available to the television match official, and may not offer a camera angle that will clearly show an incident on the field of play.

As mentioned, this system is only used at the highest level of the game, and there is no equivalent for the amateur competitions in Ireland. This means that any technology that is easily implemented and not costly could benefit the amateur game in Ireland greatly.
1.2.3  Use of Android Platform

Android provides a Java-based development platform for applications on the Android Operating System.

1.2.4  Use of OpenCV

For this project, I planned on using OpenCV, which is an open-source library used for Computer Vision processes and operations. OpenCV would allow me to perform operations at the pixel level of an image, as well as offer samples of similar projects that may have used similar methods or functions as I would use.

1.2.5  Design Camera Interface

For this project, a camera interface specific for this app needed to be designed. Android provides a certain amount of documentation of how someone can design and implement their own camera interface; however, the information becomes slightly disjointed when it comes to customising it and changing the design radically. Using Android’s “Activity” object, it is possible to recreate the interface seen in inbuilt Android Camera apps. However, as mentioned, this project needed a different design to that. My approach to this would be to use a combination of existing Android features, as well as some features included in a library I would be using.

1.2.6  Analysing Camera Frames

In order to decide which frames or sequence of frames should be recorded or kept, the current frame needed to be compared to the previous frame or frames. Once this had been done, it could be determined whether or not there had been a sufficient enough change in pixels between the sets of frames. If there was more than a predetermined amount of change in the current frame, then the frame would be recorded to a video file. Otherwise, the frames would not be used.

To do this image processing, I planned to use OpenCV4Android. This is a Computer Vision library specifically tailored for Android development and applications.
1.2.7 Storing frames as video

After analysing the camera frames, the application needed to save the current frame if it was determined to be suitable for storage. This meant that there needed to be a function or operation that would allow camera frames to be written to a video file, and then stored on the Android device. The video files would then need to be monitored so that they were deleted if they were not opened or watched after a period of time, as well as immediately deleting any files that were adjudged to be too short. The suitable videos would be stored on the Android device.

Initial research showed that OpenCV provided a class for converting images to video, called FFmpegFrameRecorder, so this class would perform the function of recording the video.

1.3 Overview of Project

In this section I will provide a brief overview of the project. I will give a top-level explanation of the main processes involved, before I go into them in-depth in Section 2.

1.3.1 Libraries and APIs used

Roughly midway through this project, I realised that some of the methods that I would need to use for this application were not available in the standard OpenCV library for Java and Android. This meant that I needed to search for a new library that offered the right functionality that my project needed. After thoroughly searching for a suitable replacement, I decided that I would use a library called JavaCV, which offered the functions that the project but had little to no documentation. Further details on these issues can be found in Section 2.
1.3.2 Application interface

The application opens with a simple camera-based interface. The majority of the screen is taken up by the camera preview, with a single start/stop button in the top left of the screen providing the user with control over the application.

![Screenshot of Application Landing Screen](image)

The start button changes to “Stop HockeyCam” once pressed, and reverts back to the original if pressed again. The screen remains on for the duration of the app, allowing the user to view the camera feed continuously throughout if so wished.

1.3.3 Camera Frame Analysis

The first step performed in the frame analysis is to detect where the semi-circle, or the “D”, is on the camera frame. Once the D is detected, the application knows where the area of interest is and can begin to compare frames. The steps taken to identify the area of interest will be discussed in Section 2.

Once the area of interest has been found, the application compares the current frame with a previous frame. The previous frame is the same frame at
the point which the D has been detected, which will be a frame where the D is empty of people or movement. This allows a good comparison between this frame and any frame with motion or activity in it. Further details will be discussed later on in the report.

1.3.4 Convert the frame to a video file

If a frame is found to be appropriate for recording and writing to video file, it is done so using a class called FFmpegFrameRecorder, which is accessed via the JavaCV library. FFmpegFrameRecorder takes a video frame and writes it to a new or existing video file (usually called “HockeyCamX.mp4”, where X denotes the video number). The time at which the video is created is also logged and stored in an ArrayList, as this allows easy additions of all the times that were to follow. The videos also contained a timestamp, which stores the length of the video in milliseconds.

1.3.5 Storing the video file

Once the video is created and at least one frame has been written to it, it can be stored to the external storage of the Android device. The video is only stored once it is at least 3 seconds long, so that only decent-length videos are stored. The application also constantly monitors how long existing videos have been stored for. If a video has not been used or moved in more than 5 minutes, then it is deleted from memory. Further details will be given in the next section.
2. **DESIGN AND IMPLEMENTATION**

2.1 **Systems, Languages and APIs used**

For this project, the initial plan was to design an Android application that would make use of the OpenCV4Android library. The basic vision of the project was to have a camera application that would be viewing a hockey pitch, and once any significant action happened within a designated part of the pitch, the camera would begin “recording”, as such, and a video would be produced containing the action.

2.1.1 **Issues with Libraries and APIs**

As discussed in the Overview, problems were discovered about halfway into the project. It was found that OpenCV for Java and Android did not provide enough functionality for this project. One of the main reasons for this was that Android and OpenCV for Android provides no function that allows you to write a live Camera frame to a video file (Google, 2012). It also does not have the ability to do certain OpenCV operations, which means that it immediately became obsolete as far as this project was concerned. This meant that an alternative would have to be found quickly.

The library that I decided to use was called JavaCV, which is an open source library available online at [https://code.google.com/p/javacv/](https://code.google.com/p/javacv/). According to the website:

> "JavaCV first provides wrappers to commonly used libraries by researchers in the field of Computer Vision."

These libraries include OpenCV, so in simple terms, JavaCV provides a Java-translated version of the C and C++ form of OpenCV.

JavaCV contains the same functions as OpenCV, but in Java. This meant that I was able to use this data in my Android application without having to worry about differing languages or lack of functionality. JavaCV also includes FFmpegFrameRecorder classes, which meant that I could successfully write images to video files in my program.
A problem with using JavaCV was that, inevitably, there was not a lot of documentation or explanation about much of the code. This is purely due to the fact that JavaCV really only translates OpenCV functions and classes to Java, so the C++ or C definitions of OpenCV elements is seen as sufficient for JavaCV. These definitions occasionally would not be very clear if you were to try and apply them to a Java project, so much of my learning for this project was going through C++ OpenCV documentation and trying to convert functions to a Java equivalent. JavaCV was also being constantly updated while this project was being completed (the latest release was published on March 3rd 2013, and included an update of the FFmpegFrameRecorder class), which made the experience of working with those libraries an interesting one. It was valuable experience working with a library that had no Javadocs or real-time “aids”, and one which enhanced the learning from this project.

The most time spent on this project was unfortunately associated with installing these libraries. Approximately 3 weeks was spent attempting to load the correct .jar files into the correct locations, as well as providing the right links in the Path of the Environment Variables of my own computer. Due to the complicated nature of JavaCV, the libraries needed to be located in exactly the right locations for Android applications, in particular, to work. This required the .so files of javacv.jar and javacpp.jar to be located in the libs/armeabi folder of the project, as well as javacv.jar and javacpp.jar to be contained within the Build Path of the project.

Despite this being done, no project worked for at least 2 weeks, as the .jar files did not load correctly. After asking on a forum about this problem, a solution was received which involved uninstalling and reinstalling every package or file associated with the project, including Eclipse and the Android SDK. Thankfully, this worked and these steps meant that projects containing JavaCV would now run, and I could move on with the project. This delay in the project was extremely frustrating and time-consuming, and resulted in some time pressure for getting the project completed on time.
2.1.2 Working with Android

When I began the project, the first element that I looked at was Android. This was the first time I had used Android, so the first 2 weeks was spent getting to grips with the Android system. Android provides a detailed template for recreating their own Camera application, with the opportunity for users to customise it to a certain extent. This camera system is based around an “Activity”. An activity is a single, focused thing that the user can do (Android, 2013), and represents all of the screens or functions of a users Android application.

As mentioned in the Overview, the Activity for creating a Camera application did unfortunately not provide enough customisation for this project. Once this had been established, I moved on and began looking for a more flexible alternative. The project would continue to use Android to host the Camera interface and environment where the app would execute.
2.2 Implementation of Project

![Diagram of Application Process Cycle]

- Android device is positioned on goal frame
- "Start HockeyCam" is pressed
- Top of the D is found using image processing, Hough Line detection and geometric operations
- Current frame and original frame are compared

[If Different]:
- Record frame to video

[If Similar]:
- Do nothing

Check previous videos' expiration times and delete if necessary

Fig. 7 – Application Process Cycle
2.3 Camera Use and Interface

2.3.1 Using the Camera to record Video

After moving away from the Android Activity approach to designing a Camera interface, a simple camera template was found in a sample Java file on the JavaCV site. The sample Java file was called RecordActivity.java (Zhang, et al., 2012), and this file also contained an example of how to record camera frames to a video file using FFmpegFrameRecorder. This file would act as a good starting point for the part of the project that would deal with recording the video frames to a file. The file provided a good, simple layout for the Camera, and this only needed to be changed slightly so that it fitted the needs of this application. The initLayout() function in the code is where the Camera interface is built, and this method is only called once throughout the application (at the beginning).

The camera interface was simple, with only one button, but this would be sufficient for the project. The button would either show “Start HockeyCam” or “Stop HockeyCam”. Pressing the button would either begin or finish recording, depending on the previous state of the application. Pressing the “Back” button on the Android device would also finish recording and subsequently exit the application.
2.3.2 Processing camera frames

Once a camera frame is retrieved from the onPreviewFrame method, it needs to be processed so that it can perform OpenCV operations. This is due to the fact that data from the Camera is stored as an array of bytes, with an NV12 pixel format. This is a YUV image with a 4:2:0 layout. It is defined as:

“A format in which all Y samples are found first in memory as an array of unsigned char with an even number of lines (...), followed immediately by an array of unsigned char containing interleaved Cb and Cr samples (...) with the same total stride as the Y samples. This is the preferred 4:2:0 pixel format.” (FourCC.org, 2013)

For the purpose of image manipulation, the image needed to be changed from YUV to RGB format. To do this, a method called decodeYUV420SP was developed. The function decodes each of the channels of the YUV image, and converts it to its RGB equivalent.

If the project was to only need a grey image, it would be simple to convert the YUV image to a grey image. This is due to the fact that the Y channel of a YUV
image is the equivalent of the grey value that you would find in a greyscale image.

2.4 Semi-Circle and Motion Detection

![Original Image](image-url)

Fig.9 – Original Image

2.4.1 Finding the D in the image

Once the image had been decoded to an RGB image, it was passed to a method that would detect where the “D” was. Once the D was found, it would serve as a reference point of where on the image should be analysed when looking for movement. This brings us back to the initial aim of the project, which was to detect movement in the action areas of the pitch – namely the D. The D would be found from one of the initial frames given by the Camera – this can be referred to as the setup phase. In the setup phase, it is presumed that:

a) The Camera or Android device is positioned in roughly the centre of the crossbar/goalpost
b) The nearest quarter of the pitch is empty of people (out of shot of the camera at least)

c) The Camera will not be intentionally moved hugely shortly after this phase

To find the D, a number of Computer Vision operations needed to be performed. The first of these was Canny Edge Detection.

2.4.1.1 Canny Edge Detection

Canny Edge detection is a mathematics-based approach to finding the edges in an image. OpenCV, and implicitly JavaCV, contain an automated function that will return an image that has had Canny edge detection applied upon it. The function requires a greyscale image as an input, as well as two threshold values as parameters. The threshold values can be described as lower and upper thresholds, whereby a pixel is either marked as an edge pixel or not, and then is also linked with neighbour pixels if the gradient of the edge (or the sharpness of the edge) is within the two thresholds.

Using cvCanny on an example image gives this result (Fig. 9 [the original image] would have been converted to greyscale before the use of cvCanny):
2.4.1.2 Dilation and Erosion of Canny Image

The next step performed on the image was to remove the noise in the image. Noise is any variation of pixels or randomness of pixels in an image, and commonly manifests itself as the form of speckles or grains in a photo or image. Figure 11 shows a common example of noise in an image. The distortion of the image, in the sky particularly, is noticeable, and needs to be removed if any accurate Computer Vision processes are to be used with the image.

![Example of Noise in an image](http://docs.gimp.org/2.6/en/images/filters/examples/noise/taj-rgb-noise.jpg)

For my image, I dilated and eroded the inputted Canny image to remove any white pixels that were not physically part of the image. Dilation and Erosion are dual processes, meaning that they do not cancel each other out, or reverse the other process, but they are in fact pretty much opposite processes. Firstly, I dilated the image once. Dilation has the effect of “filling in” any small holes in the image. I then eroded the image twice, which has the opposite effect of dilation. Erosion removes any small amounts of noise from the image, and makes pixel groups smaller. The reason for eroding the image twice was to remove as much
noise as possible, while leaving important features, especially the D, intact. The result image of the two processes being performed is shown in Figure 12 below.

![Fig.12 – Dilated and Eroded Image](image)

### 2.4.1.3 Hough Lines

The next step in detecting where the D was in the image was to use Hough Lines to find all the lines in the image. The Hough Transform is the umbrella term for a variety of transformations that can be used on images to determine whether certain features exist or not. In this case, I decided to use Hough Lines to find all the linear features in my image.

I decided to use Hough Lines for my image after previously trying to use Hough Ellipse. Hough Ellipse would ideally have been a better approach to the project as a whole if it had worked, as it would have offered more flexibility in the project had it been possible. This is because the D on the pitch warps to an ellipse when being viewed from the perspective of the goal. Hough Ellipse would have meant that the positioning of the Camera in and around the goal would not have been as
important as it is with using Hough Lines. This is due to the fact that Hough Ellipse would obviously detect any ellipses in the image, and the shape of the D would seemingly be perfect for that kind of analysis. However, with the Camera positioned as it was for this project, the top of the D was very linear, and an ellipse was not being detected using Hough Ellipse. This meant that I used Hough Lines instead.

In the notes for Hough Lines presented by Dr. Kenneth Dawson-Howe, he explains the basic formula for Hough Lines as such:

“For lines the most familiar equation is \( j = mi + c \) (usually presented as \( y = mx+c \)) where \( m \) indicates the slope of the line. Unfortunately this line equation cannot represent some lines (i.e. \( i = p \) where \( p \) is any constant) so an alternative form of line equation must be used:

\[
s = i \cos \theta + j \sin \theta
\]

where \( s \) is orthogonal distance of the line to the origin and \( \theta \) is the angle between that line and the \( I \) axis.” (Dawson-Howe, 2012)

OpenCV/JavaCV provide inbuilt methods for using Hough Lines. For my project I used `cvHoughLines2()`, which returns the start and end point of all lines found in the image. With the start and end points of the lines, you can easily plot the lines on an image. Figures 13 and 14 show the result of Hough Lines when applied to an image.
Fig. 13 – Screenshot from Camera

Fig. 14 – Hough Lines applied on fig. 13
2.4.1.4 Detecting the D

Now that all the lines in the image were found, the next step was to identify which of the lines were parts of the D. To do this, I checked each pixel along the centre of the x-axis in the image, and checked whether or not it was lying on a Hough line or not. The pixels that were to be checked are along the red line, as shown in Fig.15 below.

![Image showing line of pixels to be analysed](image.png)

Fig.15 – Image showing line of pixels to be analysed
At each pixel along the mid-line, I used the equation of a line formula to check whether that pixel was on one of the lines found using Hough Lines. The equation of a line formula is:

\[ Y = mX + C \]

Where \( m \) is the slope, and \( C \) is defined as the Y-intercept (where the line crosses the Y axis). \( M \) is calculated by:

\[ \frac{y_2 - y_1}{x_2 - x_1} \]

While the Y-intercept is calculated by:

\[ C = y_x - m_x \]

Once the equation of a line for any of the Hough Lines was found, it was possible to plug in the points along the mid-line and check if they were on that line. If they were on that line, the equation would be satisfied. Otherwise, the point is not on that line, and the program would move onto the next Hough Line.

The program worked from the bottom of the mid-line in the image upwards. This is so the first Hough Line that satisfied the equation would usually be a line that lay on the D. It can be reliably assumed that the first Hough Line found is on the D due to the image processes performed this step and that have been explained previously. Once the top of the D had been found, the program moved onto analysing and comparing frames in real time. Before it moved on, however, it saved the current frame that was being looked at in relation to finding the top of the D. This frame would serve as the comparison for all future frames that were to be analysed, as it would be relatively empty of people or movement, and would act as a good benchmark for the frame comparison.

### 2.4.2 Comparing Camera Frames

Now that the D had been found, it was possible to compare frames and only look at the area of the pitch that the application is focused on. In the function in the program that would compare frames, the current frame would be taken and
compared to the frame saved in the previous step (when the D was found). The RGB values of each pixel would then be retrieved and compared to its counterpart pixel in the comparison image. If the RGB values had changed by a certain, arbitrary value (which in the code is named \texttt{THRESHOLD} and set to 20), a counter called \texttt{numChanged} would be incremented. This variable would store the number of pixels that had changed by the required amount. A counter of the number of pixels being analysed was also kept.

Once all the pixels in the action area had been compared, the number of changed pixels (\texttt{numChanged}) was divided by the number of pixels in the area (\texttt{numPixels}). If the result of this division was greater than a certain amount, then the frame was considered to have changed enough, and it should be recorded to video. This result would mean that the method would return a true value. Otherwise, the frame had not changed sufficiently and a false statement was returned.

The amount of change needed for a frame to be considered changed enough was 0.003, or 0.3\%. This figure was obtained through thorough testing and trial-and-error. This figure was decided to be the minimum at which enough change had taken place between frames. In reality, it equated roughly to one person in the Camera frame approaching the D, and just about to enter it. This is the minimum amount of action worthy of recording that you would expect to see in a real-life hockey match, so the figure of 0.3\% sufficed.

\section{2.5 Video Storage}

\subsection{2.5.1 Storing the video}

When a frame is due to be stored as a video, it needs to be converted to a video file. This is done using the aforementioned feature of \texttt{FFmpegFrameRecorder}, which writes images to a video file. When a video starts being recorded, a timestamp is saved of the time that frame is written to video. This is done for each frame so that the length of the video can be found once the video is recorded fully.
Videos that are recorded by the application are stored as .mp4 files in the external storage directory of the Android device – which is usually the SD card. The video files are stored with the name “hockeycamX.mp4”, where X is a number. X is incremented every time a possible new video file is created, so the newest video file has the highest index. The file can then be found on the SD card or external storage device, such as the example below:

Fig.16 – Screenshot of Video File on SD card
2.5.2 Checking duration of videos

In order to save storage and make the application efficient, all of the videos have a timestamp and a finish time. The timestamp records how long the video file is in milliseconds, and from there the application can judge whether or not the video is worth keeping. If the video file is less than 3 seconds long, then it is discarded at the time of recording. This means that the storage space will not be consumed by numerous short length video clips.

Each video also has a finish time associated with it. This time is in 24-hour format, and is added into an ArrayList. This means that the key of the ArrayList corresponds to the number of the video. The reason for storing the time at which a video file finished recording is to check how long it had been in storage. If a file has been stored for more than 5 minutes, then it is adjudged to have been unused and will not be used in the future, and is therefore deleted from memory. This helps save memory on the Android device, as well as allow the application to run quicker as less of the external storage will have to be cycled through to find a free space for a new video.
2.6 Positioning of Camera

The Camera is designed to be mounted in the centre of the crossbar of the goal. This means that the application is calibrated for any slight variation on this. It is suited to any non-major elevation changes also, which gives flexibility to the user.
Fig.18 – Camera mounted on goal frame
3. **EVALUATION**

3.1 Limits and Constraints of Project

Android is a fairly new technology. The first Android smartphone went on sale in mid-late 2008 (Holson & Helft, 2008), while the first release of the Android Software Development Kit (SDK) was also launched around that time. Since then, Android has grown to be one of the two largest smartphone operating system developers in the world (along with Apple).

Because of this rapid growth, other software may have struggled to keep itself updated at the same rate as Android has been updating its own program, or indeed may be behind Android purely because they started later and further functionality is still be added to its Android customisation. The latter is the case with OpenCV4Android – which is OpenCV’s Android library. OpenCV does not provide the full amount of features or functions for its Java/Android library compared to its standard C or C++ libraries. This was an influential factor in my project, as some of the features that were integral to its execution were not accessible on Android. This meant that, as is mentioned at the start of Section 2, another library had to be found. This library – JavaCV - offered a good alternative to OpenCV, but it lacked the documentation and information that a more popular and widely-used library system, like OpenCV, would offer. In the end, JavaCV did not present any fundamental difficulties, but the ones that it did have will be outlined in the Difficulties section later in this report.

One other limitation that influenced this project was the use of a mobile phone, and in particular, the processing power that it had. Because of the computational cost of some of the functions or operations that the project used, the “real-time” analysis of the camera frames was delayed and staggered, as the program was not able to cope with the frame rate while performing all the operations necessary. This problem could be solved in two ways. Firstly, a more powerful device could be used, therefore eliminating the problem of lack of processing power. However, this could mean that the Android device could be bigger, and that could affect the physical positioning of it on the hockey goal frame as if it
was too large to fit on the crossbar, an alternative position would have to be found. The second solution to the problem could be to alter the program so that the computations and operations are more efficient and less costly on the system. More detailed explanations of this solution will be found in the Difficulties section of the report.

3.2 Successes

This application was largely successful. Many of the processes or applications used within it worked as planned, and returned favourable or correct results.

The saving of videos, as well as the monitoring of the age of the previously saved videos, was the first part of the project that worked efficiently and effectively. The saving of video as a .mp4 file was seamless, as well as the check of whether or not it was long enough to be stored. This part of the project was pleasing, especially because, as mentioned, it was the first part of the project to fall into place and work well.

The majority of the JavaCV processes that I used also performed well and correctly. Many of the steps that I planned on taking before the project worked well, and returned the exact result that I would have expected. This meant that I had confidence in trying new steps, and adding features to the program if needed.

I was also pleased with some of the approaches that were used to solve issues in the project. One such example is the use of the equation of a line formula to solve whether or not a point lay on a line. This formula worked extremely well, as well as not being expensive from a processing point of view. High-performing steps like this work towards increasing morale and happiness with the project as a whole, as well as keep interest in the work high.
3.3 Difficulties

There were three main difficulties encountered during this project. Of course, there were additional smaller issues, but the three that will be described below consumed at least one whole day of work to resolve.

The first major issue I had was installing the JavaCV libraries that I needed. As JavaCV is not a widely used or documented system of libraries, it required further installation compared to other more well-known libraries. This required the changing of System Variables on my own computer, as well as installing a selection of folders that each needed to be stored in the right locations. A number of .jar files contained within these folders also needed to be unzipped, and their contents (numbers of .so files) moved to a folder within the project. The steps for installation on Android can be found on the JavaCV home page (Audet, 2013).

The biggest issue with JavaCV, however, was its initial refusal to install and operate on my computer. At least three weeks were wasted attempting to install all the required libraries in the correct place, but it was to no avail. In the end, I was able to solve the issue thanks to a reply I got from the founder of the JavaCV library, after I had posted my issue on the end of a discussion about another question that another user had (https://code.google.com/p/javacv/issues/detail?id=104#c17). This resolved the error and meant that the project could be resumed.

The second major issue in the project was to do with the inefficiency of certain operations in Java or JavaCV that caused the application to be slow at runtime. An example of this is the technique used to find lines using the Hough Transform. This process is expensive in terms of processing power, and is the source of certain academics attempting to find ways to make it more efficient. The following three academic papers present variations of the Hough Transform in an attempt to make them less inefficient:

1) “Detection of Lines in Images by Curve Fitting Using Hough Transform”  
   (Sewisy, 2007)
2) “Use of the Hough Transformation to Detect Lines and Curves in Pictures”  
   (Duda, et al., 1972)

All three papers present alternative methods to use the Hough Transform, however none of these methods are realistically applicable for this project, as they are very maths-based, and could in fact be less efficient than the inbuilt functions of JavaCV if they are not easy to implement in Java.

The third main difficulty in this project was having to decode the information received from the camera. This issue was not extremely time consuming, but the timing of the problem was particularly awkward, as it happened only a few days before the project presentation and demonstration. After a day of attempting numerous potential solutions, a method called decodeYUV420SP was found, which converted the pixel format of the image to a format that would enable image processing. This issue was less severe than the other two mentioned previous to it, but the sheer timing of it made it no less critical.

3.4 Future Work

The area of sports analysis, both in-game and post-match, is rapidly growing and increasing all the time. Mainstream sports like football (soccer), rugby and American football all have extensive technologies used regularly. These range from in-game Television Match Official (TMO) systems, as mentioned in the Introduction, to post-match analysis tools, such as Gamebreaker (http://www.sportstec.com/software/manuals/SportsCode_Gamebreaker_V7_WEB.pdf).

Compared to mainstream sports, hockey has comparatively no financial backing, and therefore lacks much of the technology seen nowadays. This project explores the amateur game of hockey in Ireland, where there is no technology used at all. This makes this project interesting, and, at the very least, provides some
evidence that there is potential for a system similar to this that is workable and implementable.

The main issue in this project that needs addressing is the inefficiency of some operations. If these could be made to work faster, and therefore reduce any lag in the program at runtime, then this project could act as a basic immediate footage playback tool that coaches, players or umpires could use during a regular hockey match. The program could also be extended so that alternate camera views can work, depending on the location of the Android device. This could mean that there are three cameras on a goal, each focusing on a different section of the D. This approach would ensure that there are no blind spots for the camera, and all action would be recorded and analysed.

Another alteration that this program would benefit from is to make some of the methods within the program more robust, and able to withstand different use cases. This may not have been an issue if the project were able to use Hough Ellipse instead of Hough Lines during the frame analysis phase, as this would have given the project much more flexibility and scope. Changes, such as this, would benefit the project, and make it a much more complete application.
4. **CONCLUSION**

This project has been an engaging, interesting and challenging proposition throughout. The aim of the project was to design an application that would analyse footage from a hockey pitch, and return video footage immediately of any action within a certain area. On the whole, this objective was completed. The project suffers due to certain issues – some of which are design and programming errors; others are system capabilities and inherent processing limitations. But it also handles many of the processes well and efficiently, making the assessment of the project a much easier task personally.

The project contains a variety of different technologies and techniques, which made it extremely interesting, and sometimes frustrating, to work on. The general experience of working on a large-scale software project, where the entire design and implementation is your own responsibility, was also invaluable. The project itself shows that there is a lot of opportunity for Computer Vision within areas such as sport, and this project is one example of that. I feel that I have shown a reasonably good approach to solving the issue of in-game analysis for amateur sports like hockey, where the only technology available for players, coaches or umpires may in fact be a mobile phone. The application has much room for improvement and change, but I believe that it serves as a good start point for any further work in a similar area.
APPENDIX

“Big Brother is watching” – FIH, 2012. *International Hockey Federation (FIH)* [Online]
Available at: http://www.fih.ch/en/news-3812-big-brother-is-watching
[Accessed 27 March 2013]

“London Olympics the first Olympics for the video umpire referral” – Go Hockey, [Date unknown]. *Go Hockey* [Online]
Available at: http://gohockey.com/london-olympics-the-first-olympics-for-the-video-umpire-referral/
[Accessed 29 March 2013]


Available at: http://en.wikipedia.org/wiki/Umpire_(field_hockey)#Video_Umpires
[Accessed 29 March 2013]

CD – CD containing project source code is attached in the back of report
BIBLIOGRAPHY


