University of Sheffield

# Computer Vision System for a Chess-Playing Robot

Gregory Ives

*Supervisor:* Jon Barker

This report is submitted in partial fulfilment of the requirement for the degree of MComp Computer Science

*in the*

Department of Computer Science

May 1, 2019

## Declaration

Signed: Gregory Ives

Date: May 1, 2019

# Abstract

Computer vision is the field of computer science which attempts to gain an understanding of the content within a digital image or video. Since its conception in the late 1960s, it has been used for a huge range of applications, from agriculture to autonomous vehicles. The aim of this project is to design a computer vision system to be used by a chess-playing robot, capable of detecting opponents' moves and returning the best move to be played by the robot. The system should be robust to changes in perspective, lighting and other environmental factors so that it can be used in any reasonable setting.

This report researches existing computer vision systems for chess, comparing the methods they use and the constraints placed on them. New techniques are then explored, with the aim of creating a system which performs well under minimal constraints. The computer vision system is designed, implemented and evaluated in the course of this report.

# Acknowledgements

I would like to start by thanking my supervisor Jon Barker for his guidance throughout the project; his insight and experience in the field helped me to shape the project into what it is. Thank you to the University of Sheffield Chess Society for allowing me to film their chess games and for dictating every move they played so I could write them down! Lastly, I would like to thank my amazing family and friends for their continued support throughout my life and my degree.

# Contents

## List of Figures

## List of Tables

# 1 Introduction

Chess is a two-player strategy board game, played on a checkered chessboard. The game reached Europe in the 13th century and the modern rules became standardised in the 19th century. The idea of playing chess against a machine dates back to the 18th century when in 1769, the infamous chess-playing automaton The Turk was exposed as a hoax (Levitt, 2000). Since the rise of digital computation in the 1950s, it has become commonplace to play chess against a computer instead of a person: in *Programming a Computer for Playing Chess*, Shannon (1950) outlines the suitability of a computer program playing a game of chess, stating that the well-defined rules and structure of chess fit well into the digital nature of modern computers.

The prevalence and performance of chess-playing computer programs have increased dramatically with the exponential rise of computational power. In 1997, a computer beat the reigning chess world champion for the first time, in the controversial match between IBM's Deep Blue and Garry Kasparov (IBM, 2012). Chess programs can now consistently beat even the best chess players, such that human-computer matches no longer attract attention from the media.

## 1.1 Motivation

Until recently, the challenge of a chess-playing computer has not been extended into the physical domain: a computer/machine performing *all* of the functions performed by a human in a game of chess, including observing the opponent's move and physically moving pieces on the board. This project revolves around an aspect of this, namely analysing visual information of the chessboard in order to infer the state of the game, which can then be used to pick the next move which the machine will play.

To a layperson, it may seem a trivial task to determine the state of a chessboard from an image or video; however, as seen in Figure 1, factors such as lighting can make it extremely difficult to detect moves being played. In this case, the move being played is `e2e4`[1] but this cannot be seen clearly in the difference between the two frames, shown on the right.

Humans are able to perform sensory recognition very well due to an understanding of the world around them. Typically, a computer vision application is unable to take this top-down approach as it does not 'know' enough about its environment. However, within the constrained domain of a chess game, it is possible to make use of this approach, as will be shown in this project.

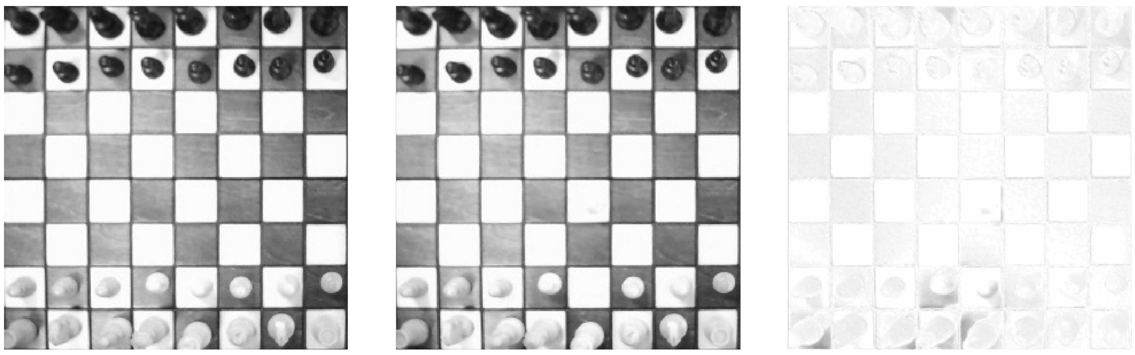


Figure 1: Chess move `e2e4`, depicted by the frame before, frame after and the difference.

[1]Move denoted by UCI notation, explained in Section 2.3.2

## 1.2 Aims and Objectives

This project aims to research and develop a robust computer vision system for a chess-playing robot, which encompasses the following challenges:

- Detect the position of the chessboard
- Detect when a move is being played and has been played
- Detect which move has been played

The focus of the project lies in the last point; given the position of the board and the time a move has been played, the system will return the move made by the player and optionally a move to be next played by the machine. Although the system is specified to be *for a chess-playing robot*, the same principals apply whether the robot is actually playing a move, or the system is just detecting moves made by two human players. If the system is to be used in conjunction with a robot, the inferred board state will be passed to a chess engine in order to instruct the robot which move to make next.

Existing implementations of chess vision systems rely on a number of constraints on the board, camera and environment; it is the aim of this project to reduce the number of constraints so that anyone can use the system with no specialist equipment, and it will still perform as accurately as in testing. In order to do this, no presumptions will be made regarding lighting, the resolution of the webcam or the chess board used.

## 1.3 Applications

The main application for this project is a chess-playing robot. However, there are many other applications which would benefit from such a system. These include,

- Automated chess game annotation: currently games are either annotated by using a specialist board, or a person writes down every move made.
- Teaching chess: as the system keeps track of the game state, it would be possible to teach someone how to play chess on the fly, for example through a connected app.

## 1.4 Report Structure

The report begins by exploring the main themes of computer vision which may be of interest for this project, namely change detection (of a similar vein to background subtraction) and object recognition or classification; existing chess vision systems will be researched, documenting their process and the constraints and outcome of the systems; essential background topics will be covered, including an insight into the technology to be used and the required understanding of chess.

Following this, the report dives deeper into the methods used to detect players' moves, and how the system will be designed and tested with some performance measure. Finally, the outcome of the system is evaluated and conclusions are drawn as to its success.

# 2 Literature Review

This chapter begins by discussing relevant computer vision techniques which can be applied to the problem. Computer vision tools are compared, as well as the available chess libraries, engines and interfaces which will bring the project together. Finally, existing computer vision systems designed for chess games will be researched, to understand their use of computer vision to determine the state of a chessboard.

## 2.1 Applicable Computer Vision Techniques

A computer vision system aims to mimic the human visual system in the form of a computational model and to perform some of the tasks which the human visual system can perform (Yang, 2009). Two approaches will be explored for determining the state of the chessboard, either:

1. Detect which cells have changed, and hence which move has been made, or
2. Classify each piece currently on the board, without necessarily using prior knowledge.

Two computer vision techniques are applicable to these approaches: change detection and object recognition.

### 2.1.1 Change Detection

As defined by Computer Vision LAB (2013), change detection is the detection of "significant" change, where a "significant" change corresponds to variation in the imaged scene's geometry. Change detection can be applied to both still photos taken over time, or to video; real-life applications of change detection range from video surveillance to submarine environment mapping (Bouwmans and Garcia-Garcia, 2019).

However, often only a small subset of the changes detected by a computer vision system can be attributed to a change in the scene's geometry. Possible disturbance factors which may affect real-world applications include:

- Image noise
- Scene illumination changes:
  - Global illumination changes e.g. a light being turned on
  - Local illumination changes e.g. shadows cast by the observed objects

Image noise is random variation in brightness or colour information of an image, and is usually an aspect of electronic noise, produced by the sensor or circuitry of a digital camera (Farooque and Rohankar, 2013).

Image noise is problematic for many computer vision systems, and denoising techniques such as linear smoothing filters are employed to reduce its impact. However, scene illumination changes will pose a larger problem for a chess vision system; whereas image noise only causes a slight variation in pixel intensity, factors such as a person walking past the chess game or a light being turned on would cause a dramatic change across a large proportion of the chessboard. The most simple form of change detection looks solely at the changes in pixels between images. However this technique is heavily influenced by factors such as lighting, so other features are often used such as edges, contours, corners and other prominent points in the image.

### 2.1.2 Object Recognition

Object recognition is a computer vision technique concerned with identifying objects in an image, from a set of known labels (MathWorks, 2019). Humans have the ability to observe an object and immediately infer the identity or category of the object, regardless of variation in appearance due to position, lighting and other factors. From an early age, children are able to generalize an object based upon its characteristic features: for example, after seeing a few examples of a chair, children would easily be able to distinguish a chair from a table (Yang, 2009).

Although a trivial task for humans, this represents an incredibly complex problem for computers. Some of the major challenges found in object recognition are similar to those found in change detection. Jain, Kasturi and Schunck (1995) define these as:

- Varying illumination
- Perspective of the object due to viewpoint
- Object deformation e.g. how large the object is
- Occlusion of the primary object

As well as these, an important factor for any classification system (but especially relevant for a chess vision system) is intra-class or within class variance. This is the variance that occurs within a class of the same object; for example, chess sets may differ dramatically in appearance, from the board itself to the pieces, but represent exactly the same game.

A straightforward comparison between a reference image and the image to be classified would not work because of these factors. Therefore certain features must be extracted from the images which are invariant to these changes. These features may be general to an object, for example, the average intensity of the object, or may refer to local features on an object; for example, the presence of an edge at a certain point.

## 2.2 Computer Vision Tools

There are many computer vision tools suited to a project of this ilk. Three viable tools were researched:

- OpenCV Python
- OpenCV C++
- MATLAB's native libraries

Each tool and programming language has its advantages and disadvantages. MATLAB is a powerful matrix library by nature, whereas Python and C++ would require libraries such as numpy (for Python) in order to provide fast algorithms for large matrix manipulation. Python, however, has become the predominant language for scientific computing; Python libraries such as OpenCV, dlib, numpy, scipy, scikit-learn and matplotlib provide a powerful environment for learning and experimenting with computer vision and machine learning.

Given the author's prior experience in Python, the project will be undertaken using predominantly OpenCV in Python, with additional libraries used as needed, for example, scikit-learn for object classification. The functionality OpenCV provides which may be useful for this project includes Canny edge detection, finding chessboard corners[2], Hough line transform and other methods for feature extraction.

[2]Commonly used for camera calibration

## 2.3 Chess Software

Since the advent of computers, the idea of playing chess against a computer has become increasingly popular. This has led to a wide range of chess libraries and engines, in order to support computer chess programs written in any language.

### 2.3.1 Chess Libraries and Engines for Python

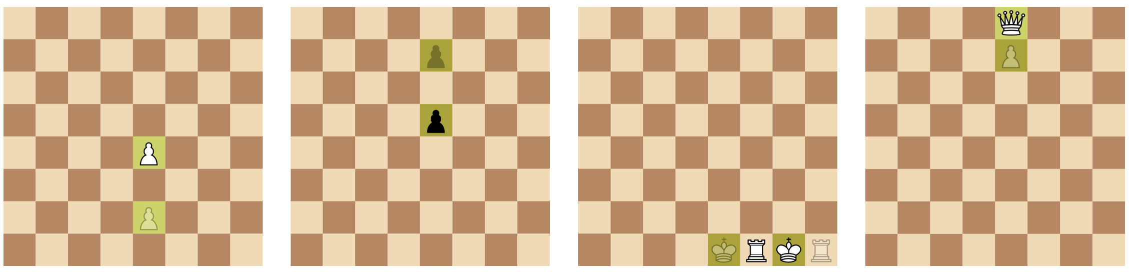
This project will require a chess library in order to assess the possible legal moves which a player can make. The most obvious choice for Python is the python-chess library, distributed in the Python Package Index. As described by its creator, python-chess is a “pure Python chess library with move generation, move validation and support for common formats” (Fiekas, 2019). The project is well-supported by its maintainers and is available open source on GitHub. In particular, python-chess will be useful for maintaining the state of chess games, and for listing the legal moves a player can make.

In addition to the python-chess library, a chess engine will be used in order to make moves on behalf of the chess-playing robot. A chess engine is a computer program which analyses the board and decides on the best move to make. The top chess engine in the world is Stockfish[3], according to the Computer Chess Rating Lists (2019). Stockfish is an open source project with an interface for Windows, Mac, Android and Linux. Although it does not explicitly provide an interface for Python, two third-party Python packages have been created for this purpose. If these packages do not function as intended, it is possible to interface with Stockfish via the standard input and output streams.

### 2.3.2 Universal Chess Interface

There are many ways to record the moves of a chess game, including algebraic chess notation, descriptive chess notation, ICCF numeric notation, Smith notation and coordinate notation. Algebraic chess notation is the official notation of FIDE which must be used in all recognized international competition involving human players (Schiller, 2003).

Algebraic chess notation is used to describe the moves made in a chess game using a coordinate system to uniquely identify each square on the chessboard. The columns on the chessboard, called files, are labelled `a` to `h` from the white’s left to right. The rows, called ranks are numbered `1` to `8` starting from the white’s side of the board.



(a) `e2e4` (b) `e7e5` (c) `e1g1` (d) `e7e8q`

Figure 2: Four examples of UCI notation of chess moves.

[3]https://stockfishchess.org/

A variant of algebraic notation is *long* algebraic notation, which specifies the starting and ending square separated by a hyphen, e.g. `e2-e4`. A form of long algebraic notation (albeit without the hyphens) is used by the Universal Chess Interface (UCI) (Kahlen, 2004). Released in November 2004, the UCI is an open communication protocol that enables chess programs to communicate, often between a graphical interface and a chess engine. Examples of UCI notation can be seen in Figure 2, where the faded pieces illustrate the previous position of the piece.

## 2.4 Existing Computer Vision Systems for Chess

Koray and Sümer (2016) present a real-time system that detected moves played in a chess game, using MATLAB. Firstly, their system detects the chessboard using the detectCheckerboardPoints function of MATLAB, before applying geometric rectification to map the board onto a square 8 by 8 grid. Their system relies on the orientation of the board, i.e. pieces being on the left- and right-hand sides of the camera. After this, Koray and Sümer propose an automatic camera exposure algorithm that aims to find the optimum exposure level which maximizes the average of the colour differences between light/dark piece and square. After applying this algorithm and other adjustments, the average colours of pieces and squares are taken as a reference for subsequent processing; their approach classifies four reference colours, namely

1. Black square
2. White square
3. Black piece
4. White piece

The implementation of move detection is based on the comparison between a reference image of the board (from which the average colour values were obtained) and a snapshot of the board throughout the game. A region of interest (ROI), defined as 25px by 25px in the centre of each square, is used to determine what colour the piece is. The average colour in the ROI of each square on the chessboard is calculated, and these values are compared by means of Euclidean distance in Lab colour space[4] in order to classify them into one of the four classes. The chessboard state of the last snapshot is obtained; this is then compared to the previous chessboard state in order to deduce which move has been played.

The system designed by Koray and Sümer works well, successfully detecting 162 out of 164 moves, over three games in different illumination conditions. However, the system is heavily constrained by the position of the camera, the orientation of the board, the chess set used, and the illumination of the environment. As stated in their discussion,

> The combination of the lighting, camera settings and chess set are playing a big role in the success of detecting moves in a chess game. Although the proposed system works well under different illumination conditions, lighting environments (having a single light source) that cast strong shadows over the board are unsuitable for tracking.

In a similar paper, Sokic and Ahic-Djokic (2008) discuss the design for a simple low-cost computer vision system for a chess-playing robot. Their proposed design is similar to that of Koray and Sümer, however, it uses edge detection as a means to calculate if a cell is occupied or not. Sokic and Ahic-Djokic's experiments with edge detection on several frames showed that edge detection processed frames are less sensitive to shadow and environment light changes, which reduced error rates in move detection. The success rate of chess move recognition was up to 99%, in good lighting conditions. Lower rates (down to 75%), were due to low camera quality, low brightness, and "exceptionally unfavourable" shadows.

[4]CIELAB colour space, also known as CIE L*a*b* or sometimes abbreviated as simply "Lab" colour space

# 3 Requirements and Analysis

The goal of this project is to create a computer vision system which can correctly identify the moves played in a chess game. As previously discussed, a computer vision system for a chess-playing robot must be able to carry out the following tasks:

- Locate the position of the chessboard
- Detect when a move is being played, and has been played
- Classify which move has been played

Although the ideal implementation would be able to complete all of these tasks fully autonomously, none of the tasks are trivial. Therefore, this project will focus on the latter: deciding which move has been played. The system should be wrapped in some form of graphical user interface (GUI), so it is intuitive and usable for both testing and demonstrative purposes.

## 3.1 Finding and Rectifying the Board

Instead of automatically identifying the corners of the board, this task can be side-stepped by manually inputting the coordinates of each corner of the board. To ensure this isn't a cumbersome task for the user, the GUI should allow the user to click each corner of the board, and the system should calculate the corresponding raw coordinates. The system should then rectify the board, by mapping the chosen coordinates onto a square 8 by 8 grid.

There are downsides to manually inputting the location of the chessboard: if the board moves during the game, this will not be accounted for by the system, which could potentially impact the move detection. If this problem becomes apparent during testing, it would be possible to mitigate this by tracking the corners of the board after they have been inputted.

## 3.2 Detecting When Moves Have Been Played

It is very common for chess players to use a chess clock, which measures the time each player has taken to move. The United States Chess Federation (2015) provides an explanation of a chess clock:

> A chess clock is actually two clocks! When you're thinking, your clock ticks down. After making a move, you hit a button at the top of the clock and your opponent's clock starts ticking... There are two main types, the digital and analog clock.

If the players are using a chess clock, it would be a simple task to set a trigger between the clock and the computer vision system; when a player hits their clock, the system knows that player has finished their move. Therefore, this project will expose a trigger (for example, an API[5] call) to indicate that a player has made a move. This project will not implement the interface between the chess clock and the computer (and the use of additional hardware should not be assumed), so will replace this with a button in the user interface, which will call the trigger in the API.

If the timestamps of each move in the game are known, it is then possible to build a sequence of keyframes, almost like a storyboard, showing the progress of the game as moves are made. This provides a layer of abstraction away from the raw video, allowing the system to make decisions based purely on consecutive keyframes.

[5] Application Programming Interface

## 3.3 Classifying Which Moves Have Been Played

After a player has had their turn, the system should be able to detect which move they have played. This should use some combination of change detection and object classification. In conjunction with these techniques, the system will interface with the python-chess chess library – by passing the state of the game into the library, the library can return the legal moves which will be used to infer what move has been played.

### 3.3.1 Classifying Moves Using Change Detection

If the initial state of the chessboard is known, the individual pieces do not need to be identified. In order to use a change detection-based approach, the raw video must first be split into frames, as discussed in Section 3.2. After a move has been played, the subsequent state of the board can be found by observing which cells have changed since the previously recorded keyframe.

The most obvious method would be to take the absolute difference between consecutive keyframes, $|\text{keyframe}_n - \text{keyframe}_{n-1}|$. However, as seen in Figure 1, the difference between frames can be very subtle and is heavily influenced by changes in lighting, such as someone moving next to the board. Therefore, the cells which have changed the most do not necessarily represent the move made.

Other comparative measures between keyframes will be explored, such as Canny edge detection, and measurements in colour space as seen in Koray and Sümer (2016).

### 3.3.2 Classifying Moves Using Object Classification

A naïve and expensive solution to an *object classification*-based approach would be to train a classifier to recognise every chess piece from the webcam above the board. However, this would require the classifier to be trained on lots of training data, across many different chess sets (otherwise this would constrain the user to a particular chess set). Despite chess pieces being easily recognised by humans, the visual cues used to identify a piece are often subtle – for example, a bishop is often recognised by the slit in the head of the piece. From overhead, it is especially hard to determine which piece is which, as can be seen in Figure 3.

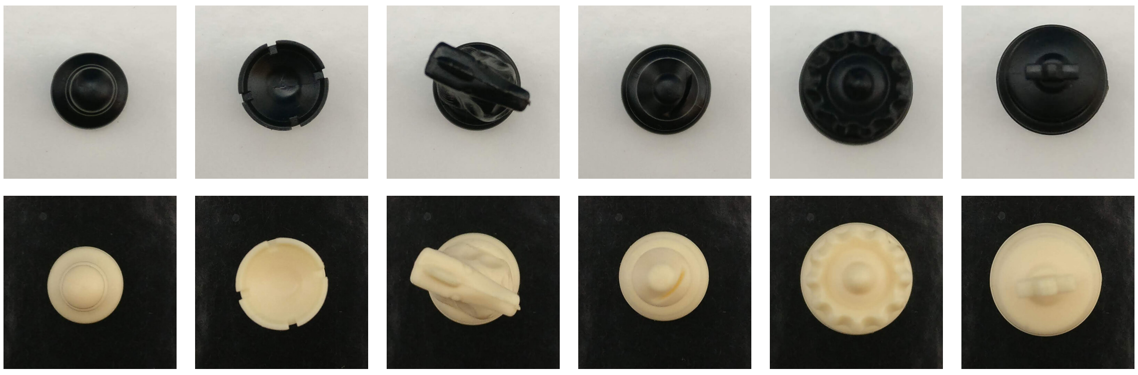

(a) Pawn (b) Rook (c) Knight (d) Bishop (e) Queen (f) King

Figure 3: White and black chess pieces from overhead.

For this reason, it would be infeasible to train a classifier to identify individual pieces without a large quantity and variety of training data. A more reasonable solution would be one which could be used in conjunction with change detection: using object classification to identify only the colour of the square and the piece within the square. Each square of the chessboard would be classified into one of the following six classes:

1. Empty black square
2. Empty white square
3. Black piece on black square
4. Black piece on white square
5. White piece on black square
6. White piece on white square

In the starting frame of a chess game, there are 64 squares, made up of 16 of each empty square and 8 of the latter four classes. This is enough information to train the chosen classifier. A simple classifier to use would be a nearest neighbours algorithm, however, if this does not produce satisfactory results, other algorithms implemented by the scikit-learn python library will be tested.

#### 3.3.3 Detecting the Players' Hands

An alternative approach to the problem is the possibility of tracking the players' hands throughout the game. The position of each player's hand can be used to deduce roughly where the move was played, for example by applying a binary mask to where the player's hand has been. As it is not possible for a player to make a move without holding the piece, the only available moves are those in which both squares are contained within the binary mask. The length of time that a player's hand rests over a position may also be indicative of the move made.

Although this solution may yield good results, it is the last resort for the implementation of this system; whereas the previous methods discussed only require certain frames from the raw video, this method would require every video frame. This is likely to make the system harder to test over lots of data and consequently less robust to changes to the board or environment.

## 3.4 Data Collection and Annotation

In order to successfully evaluate the performance of the system, a large amount of video data must be available. Two options are available for gathering this data: either manually film games of chess or simulate games using 3D modelling software, such as Blender[6].

#### 3.4.1 Manually Filming Chess Games

Collecting the necessary amount of data manually would require filming hours of chess games. Despite being a tedious task, this would produce the most relevant data, as it is exactly what the system is designed for.

In order to speed up this task, ethics approval has been obtained to film others playing chess; this will allow the author to approach the University of Sheffield Chess Society, who hold weekly meetings, in order to obtain more data which can be used to train and/or test the system. The information sheet and user consent form can be found in Appendices A and B respectively. The ethics application and approval letter can be found in Appendices C and D.

[6]https://www.blender.org

If the video data is collected manually, the data will also need to be annotated manually. It may be possible to speed this up by creating a section of the GUI specifically for annotating games of chess, with an easy to use interface and the option to speed up video data.

### 3.4.2 Simulating Chess Games

If filming chess games manually becomes too arduous, or an unexpectedly large amount of footage is needed to train the system, it may be viable to simulate videos of chess games. Using 3D modelling software, such as Blender, it would be possible to render thousands of keyframes which the system could be trained upon; the problem would be modelling the chessboard to be as realistic as possible.

If the simulated videos were not representative of real footage, there would be a possibility of overfitting the system to the training data. This is explained in further detail in Section 3.5.2.

## 3.5 Testing and Evaluation

In order to determine the success of the computer vision system, it must be evaluated comprehensively in various situations. The first decision to be made is the evaluation measure – what measure will best evaluate the accuracy of the system and can be used to compare different configurations or versions. Configurable parameters and testing sets are discussed, as well as how to test the system's limits.

### 3.5.1 Performance Measure

The most obvious performance measure is the percentage of correctly identified moves across a number of chess games. However, if the system incorrectly identifies a move towards the beginning of a game, this would negatively impact its perceived performance throughout the rest of that game. This would be a poor performance measure, as it is dependent on when a move occurs in the game. Instead of measuring the accuracy across all moves in the game, the system must evaluate each move individually throughout the game; after the move has been evaluated, the system must then correct the move, in order to avoid the incorrect classification of further moves. This performance measure will accurately assess the system's performance over a game, independent of when a move is incorrectly identified.

### 3.5.2 Parameters and Testing Sets

The computer vision system will be based on configurable parameters, which will be tuned to find the optimum solution. When tuning parameters, it is possible to *overfit* the parameters to the data which the system is being trained upon: although the accuracy of the system may seem to be improving when applied to the training set of data, this may not be the case when it is applied to new, unseen data. In order to prevent overfitting when tuning the parameters of the system, K-fold cross-validation will be employed.

The following K-fold cross-validation method will be used.

1. Split the data into K sets.
2. For each unique set:
    1. Hold out the set for testing.
    2. Take the remaining sets as the training set.
    3. Find the optimum parameters for the training set.
    4. Evaluate the parameters on the test set.
    5. Retain the evaluation score and discard the parameters.
3. The final evaluation score can be found by the average of the K intermediate evaluation scores.

### 3.5.3 Testing the System's Limits

The system should be tested thoroughly to find the limits of its capability, an indication of how robust the system is. This will require recording video of chess games outside of the 'reasonable' boundaries, for example from extreme angles. Therefore the collected data must include videos which are varied in respect to:

- Angle of webcam in the vertical plane
- Angle of webcam in the horizontal plane
- Lighting conditions

The overall performance of the system will be determined by its accuracy on the subset on videos which represent a 'reasonable' scenario, one which has good lighting and where the camera is pointing down at the board from near vertical. The system will also be tested on the subset of videos which do not constitute a reasonable scenario, in order to test its limits.

# 4 Design

This chapter covers the design of the system, to match the requirements of the system as specified in Chapter 3. First, an overview of the application structure is presented, before diving deeper into each respective part of the system.

## 4.1 Application Structure

In order for the system to be useful to a wide audience, the system should be platform-agnostic[7]. Therefore, this project will be implemented as a web application, sat upon the Python back-end; the back-end of the system will comprise the computer vision system and a server. The front-end of the system will be created using a front-end framework, discussed in Section 4.4.

Figure 4 shows the communication between the components of the system. As shown by the API calls, the system will be loosely-coupled between the front-end and the back-end; this is important because it enables the portability of the system to another platform[8].

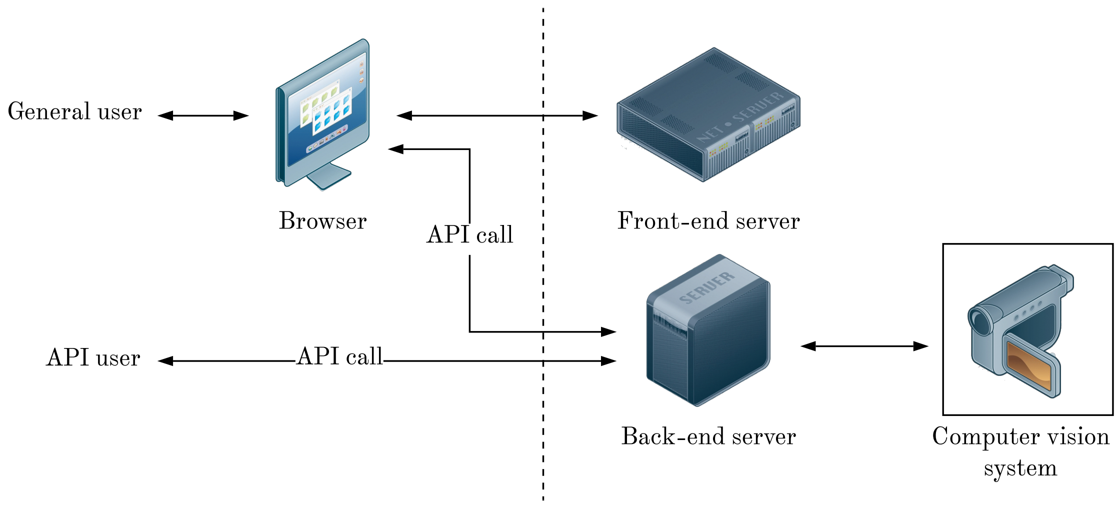


Figure 4: Diagram showing the structure of the application.

## 4.2 Computer Vision System

The main part of the system, the computer vision system should be able to take raw video from a webcam and output the game state. So that the computer vision system is modular, it will be written in the form of a Python package.

A python package typically has the structure shown in Figure 5. The `setup.py` file sets up the package ready to be imported by other applications; the `__init__.py` file is the entry module to the package, where classes will be exported. Finally, there exist Python modules (in this case only one, `module.py`) which contain the main code of the system.

[7]Works on any operating system (within reason)

[8]A new application could be built for the system without changing any of the back-end code

```
package/
    package/
        __init__.py
        module.py
    setup.py
```

Figure 5: Typical structure of a Python package (Torborg, 2019).

It is expected that this project will require a fairly large amount of code, so the project will be designed using the object-oriented programming (OOP) paradigm. This will ensure the program is modular and maintainable. In order to ascertain the various classes needed, first we must step through the basic approach:

1. Load the given webcam device or video file.
2. Crop the video to the chessboard.
3. Continuously read frames from the webcam or file.
4. When a move is made, save the current frame.
5. Compare the last two saved keyframes, to find the move made.
6. Return the move made, and push it to the chess state.

Reading frames from the webcam or file should be a threaded operation[9], so the program can both read frames and process frames at the same time; this will ensure that the video is processed in real-time. Care must be taken to write *thread-safe* code, meaning problems will not arise if multiple threads attempt to mutate a data structure at the same time. As the Python classes in this program will share the video frames across threads, the frames will be stored in a 'frame buffer', implemented by a Python List data structure.

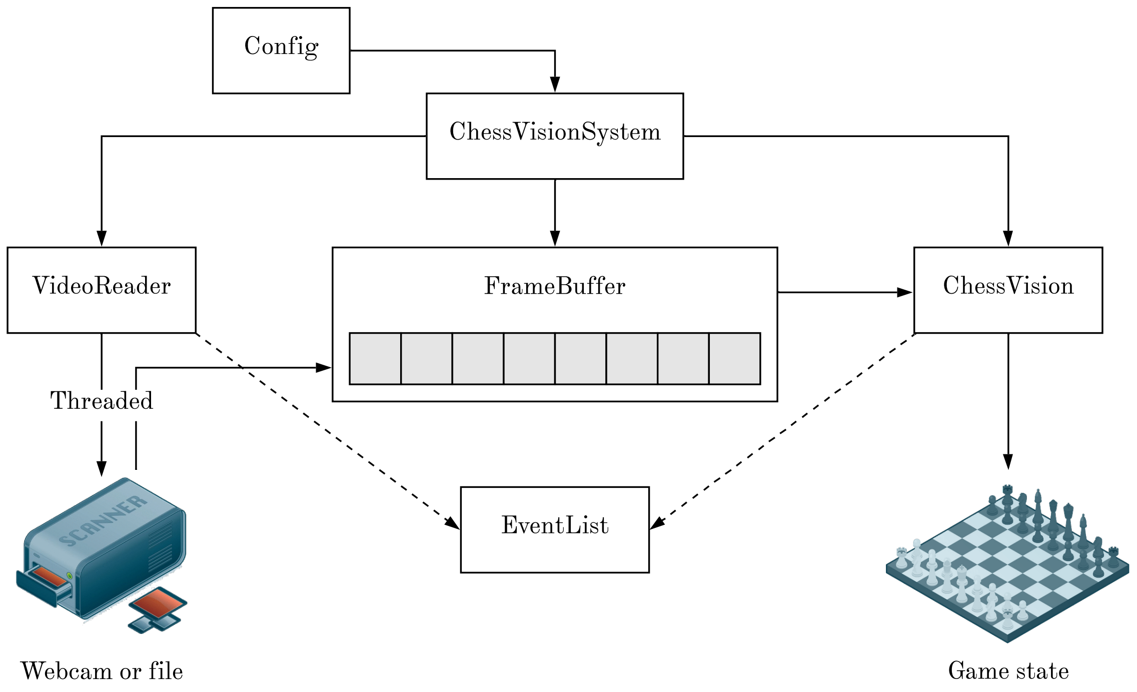


Figure 6: ChessVisionSystem class interaction.

[9]An activity that is run in a separate thread of control (Python Software Foundation, 2019)

The class structure of the Python package will be implemented as shown in Figure 6. The main class of the system will be the `ChessVisionSystem` class; in this application, the server will instantiate the computer vision system by creating an object of this class. It can optionally take a Config object, which will be used to test the system over multiple configurations. The `ChessVisionSystem` will then create three new objects: a `VideoReader`, which will read frames from the webcam or file in a new thread; a `FrameBuffer`, used to store the read frames; and a `ChessVision` object, which will compare the two previously saved keyframes and push the move made to the game state. An `EventList` object will be referenced by the `VideoReader` and `ChessVision` objects, to store the events which happen during runtime, for example, loading and cropping the video, and recording which moves occur.

## 4.3 Back-end Server

As the computer vision system will be made using Python, it would make sense to also use Python for the back-end server. There are two popular frameworks for a Python-based server application: Django and Flask. Django is a fully-fledged MVC framework "that encourages rapid development and clean, pragmatic design. Django takes care of user authentication, content administration, site maps, RSS feeds, and many more tasks – right out of the box" (Django Software Foundation, 2019). Django is aimed at larger applications, with the need for all of the out-of-the-box features it provides.

Flask, on the other hand, is a "microframework" primarily aimed at smaller applications with simpler requirements. As explained by the creators of Flask, the Pallets Team (2010),

> "Micro" does not mean that your whole web application has to fit into a single Python file, nor does it mean that Flask is lacking in functionality. The "micro" in microframework means Flask aims to keep the core simple but extensible.

This makes Flask an ideal framework for a project of this ilk. The Flask server will act as an API server, ensuring that the back- and front-end of the system can act independently.

### 4.3.1 Application Programming Interface

An application programming interface provides all the building blocks needed to create an application, which can then be combined when creating the user interface; APIs are commonly provided by both native applications and most server-based applications.

There are many protocol specifications for an interface of this kind, including SOAP[10] and REST[11]. RESTful APIs are the most common form, provided by the majority of technology products, for example, Facebook, Twitter, Google products (e.g. Google Maps) and GitHub. Responses from a REST API can be in the form of HTML, XML or JSON; an example of the latter, from GitHub's JSON REST API, can be seen in Figure 7.

[10]Simple Object Access Protocol
[11]Representational State Transfer

```
// API call to https://api.github.com/users/gregives
{
  "login": "gregives",
  "id": 23280125,
  "avatar_url": "https://avatars0.githubusercontent.com/u/23280125?v=4",
  "type": "User",
  "site_admin": false,
  "name": "Greg Ives",
  "company": null,
  "blog": "https://gregives.co.uk",
  "location": "Sheffield, UK",
  "public_repos": 2,
  "public_gists": 0,
  "followers": 18,
  "following": 14,
  "created_at": "2016-11-05T15:27:23Z",
  "updated_at": "2019-04-18T17:02:37Z"
}
```

Figure 7: An extract of JSON response from GitHub's REST API (api.github.com).

The back-end server of this application will expose a REST API which will control the computer vision system. The various API endpoints will be able to control video playback, for example playing and pausing the video, cropping the video to the chessboard, and an endpoint to indicate when a move has been played. These will adhere to the typical REST architecture, as shown in Table 1.

| Resource | Method | Description |
|---|---|---|
| `/devices/` | GET | Return the available webcam devices |
| `/videos/` | GET | Return the uploaded videos |
| `/videos/` | POST | Upload a new video file |
| `/videos/:video/load` | POST | Load the video with the given configuration |
| `/videos/:video/pause` | POST | Pause or unpause the video |
| `/videos/:video/crop` | POST | Crop the video to the given coordinates |
| `/videos/:video/move` | POST | Return the move just played |
| `/videos/:video/stream/:type` | GET | Return the video stream of the given type |
| `/videos/:video/frame/:type` | GET | Return a single video frame of the given type |

Table 1: Design of the REST API, showing endpoints and their function.

## 4.4 Front-end Graphical User Interface

The front-end server, shown in Figure 4, will serve a reactive web application to the client. There are many JavaScript frameworks available to build web applications effectively, such as React[12], Angular[13] and Vue[14]. Vue is a perfect choice for this application as it is very versatile, allowing you to start with only the basic parts of a Vue application, and include new features as the project develops; as explained on Vue's Introduction page (2019),

> Vue (pronounced /vju:/, like view) is a progressive framework for building user interfaces. Unlike other monolithic frameworks, Vue is designed from the ground up to be incrementally adoptable.

In conjunction with Vue, the front-end of the application will be created with Bootstrap, the most popular HTML, CSS, and JS library in the world (Bootstrap Team, 2019). This will enable the application to be created quickly using Bootstrap components, whilst maintaining good user experience and accessibility. BootstrapVue[15] provides a comprehensive implementation of Bootstrap components for Vue, without Bootstrap's original dependency on jQuery.

### 4.4.1 Prototypes

Firstly, an initial rough prototype was created to get a feel for the dashboard layout, shown in Figure 8. The dashboard includes 3 'panes': the leftmost pane displaying the live video feed of the chessboard, the middle pane showing the various filters which will be applied to the video (e.g. difference between frames, edge detection) and finally the rightmost pane which will show the output of the computer vision system, the state of the chess game. Controls for the video playback will lay immediately below the video feed.

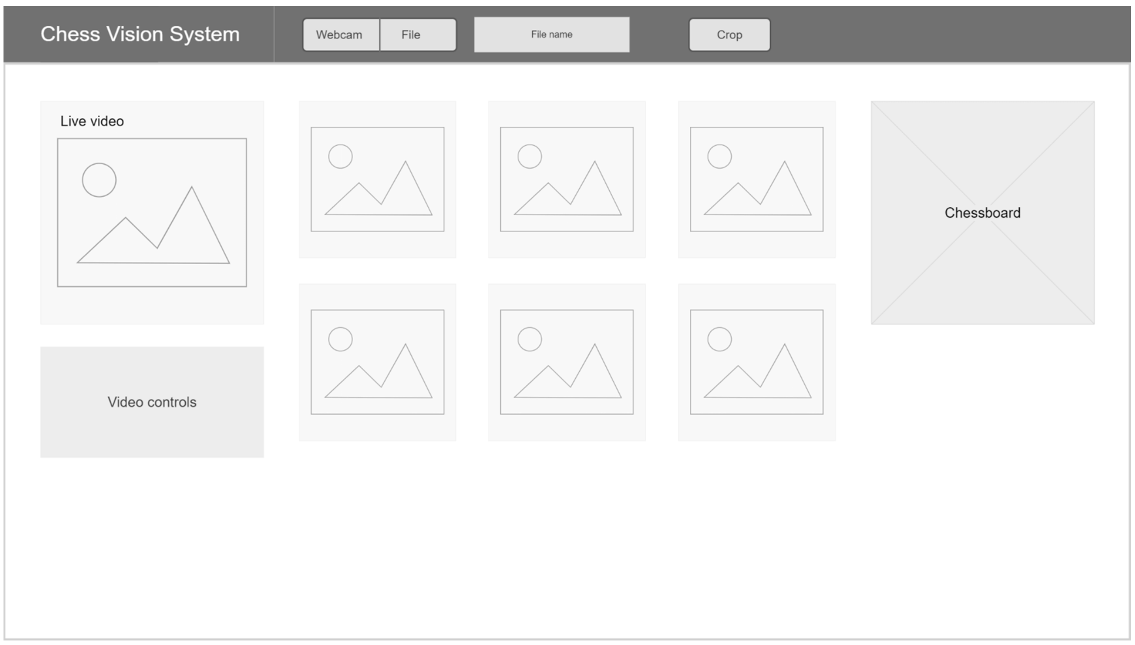


Figure 8: Initial rough prototype of the web application.

[12]https://reactjs.org
[13]https://angular.io
[14]https://vuejs.org
[15]https://bootstrap-vue.js.org

Following the initial rough prototype, the dashboard has been implemented using HTML and CSS with the Bootstrap framework, as shown in Figure 9. This gives a firmer idea of what the final user interface will feel like and allows preliminary connection with the API server. As the project continues, the dashboard will likely evolve with it, to incorporate new features and improve its usability, but the initial theme will be retained.

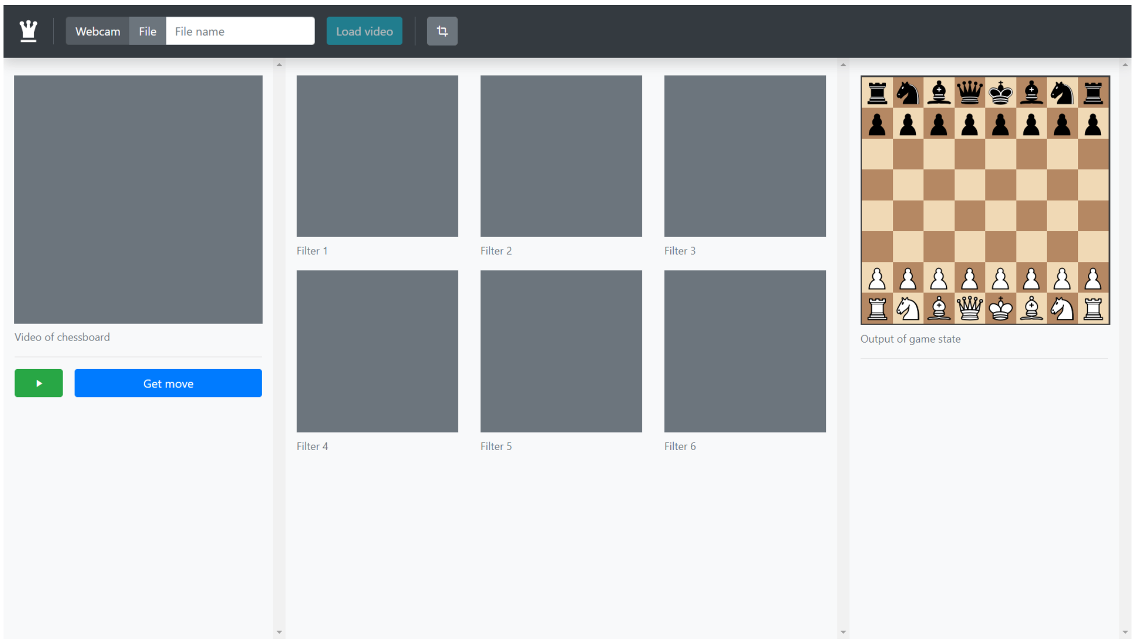


Figure 9: Prototype of the web application created with Bootstrap.

### 4.4.2 Live Video Streaming

Essential to this web application is the ability to stream video from the back-end server to the front-end. A blog post, *Video Streaming with Flask* (Grinberg, 2014), shows a proof of concept, however key to this project is the integration of OpenCV. The example code in the blog post was adapted to read video data from the webcam using OpenCV; the minimal amount of code to achieve this can be seen in Appendix E.

# 5 Implementation and Testing

Having discussed the requirements and design of the system in the previous chapters, this chapter discusses the final implementation of the system. The computer vision system is presented, screenshots are provided of the graphical user interface, the final API server is documented and the evaluation of the system is discussed.

## 5.1 Computer Vision System

The computer vision system employs both edge detection and the K-nearest neighbours algorithm for classifying which move has been made. The function used to detect which move has been made is as follows:

1. Retrieve the two previously saved frames, keyframe$_n$ and keyframe$_{n-1}$.
2. Apply the K-nearest neighbours algorithm to keyframe$_n$ and retain the resulting probabilities. For each square of the chessboard:
    i. Find the average intensity and variance of the square, optionally applying padding to the edges of the square. The average intensity and variance denote the square's coordinates in feature space.
    ii. Take the 8 nearest neighbours of the square in feature space, using Euclidean distance as the distance metric.
    iii. The probability of the square being of a certain class is proportional to the number of neighbours of that class. For example, if 4/8 neighbours are an empty white square, the current square has a 0.5 probability of also being an empty white square.
3. Apply the Canny edge detection algorithm to the chessboard. For each square, find the difference between the number of edges in keyframe$_n$ and keyframe$_{n-1}$.
4. Find the probability of each possible move occurring. For each possible legal move:
    i. Find the probability of the move according to the K-nearest neighbours algorithm.
        a. Find the probabilities of each square being correct using the probabilities calculated in step 2.
        b. Average the probabilities of each square to find the probability of the whole chessboard being correctly classified by the K-nearest neighbours algorithm.
    ii. Find the probability of the move according to the Canny edge detection algorithm. Take the absolute difference of the square which the piece is moving into, and negate from this the difference of the square which the piece has moved from.
    iii. Multiply the two previous probabilities for the move, as determined by the K-nearest neighbours algorithm and by Canny edge detection.
5. Return the move with the highest probability.

## 5.2 Graphical User Interface

The user interface, in the form of a web application, has been implemented with Vue and BootstrapVue. It acts as a client of the back-end API, rather than a tightly-coupled application. The interface provides a dashboard comprising of three pages: a demonstration page, an annotation page and a testing page. The user can select the desired page in the navigation bar.

On the left of all pages, the user can select the video file(s) or webcam device which they wish to run the computer vision system on. The design has remained mostly the same as that discussed in Section 4.4.1, with a three-pane layout on the right of the file picker. Extra features have been added such as greater control of video playback and the pages for annotation and testing.

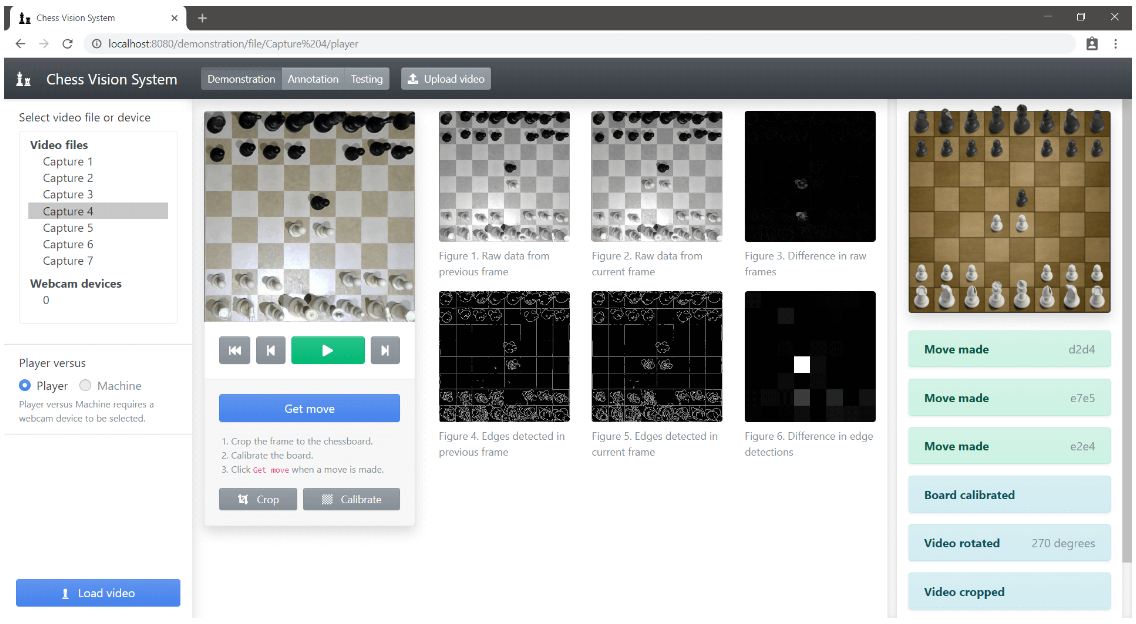
### 5.2.1 Demonstration Page

Shown in Figure 10, the demonstration page outlines the process of the computer vision system. The leftmost pane includes a video playback component, with a live video feed of the chessboard and several controls. An explanation of the process is explained to the user:

1. Crop the frame to the chessboard.
2. Calibrate the board.
3. Click `Get move` when a move is made.

The central pane shows the filters applied to the keyframes. The first row of images shows the raw video in grayscale, with the previous frame, current frame, and the absolute difference of the two. The second row shows the frames with Canny edge detection applied to them.

The rightmost pane of the demonstration page shows the outputted state from the computer vision system. An event list displays information such as when the video has been loaded and cropped, as well as the moves which have been classified. Above the event list is the chess state, displayed using chessboard.js[16], a standalone JavaScript chessboard.


Chess Vision System
localhost:8080/demonstration/file/Capture%204/player
Chess Vision System
Demonstration
Annotation
Testing
Upload video
Select video file or device
Video files
Capture 1
Capture 2
Capture 3
Capture 4
Capture 5
Capture 6
Capture 7
Webcam devices
0
Player versus
Player
Machine
Load video
Get move
Crop
Calibrate
Figure 1. Raw data from previous frame
Figure 2. Raw data from current frame
Figure 3. Difference in raw frames
Figure 4. Edges detected in previous frame
Figure 5. Edges detected in current frame
Figure 6. Difference in edge detections

Player versus Machine requires a webcam device to be selected.

1. Crop the frame to the chessboard.
2. Calibrate the board.
3. Click `Get move` when a move is made.

Move made d2d4

Move made e7e5

Move made e2e4

Board calibrated

Video rotated 270 degrees

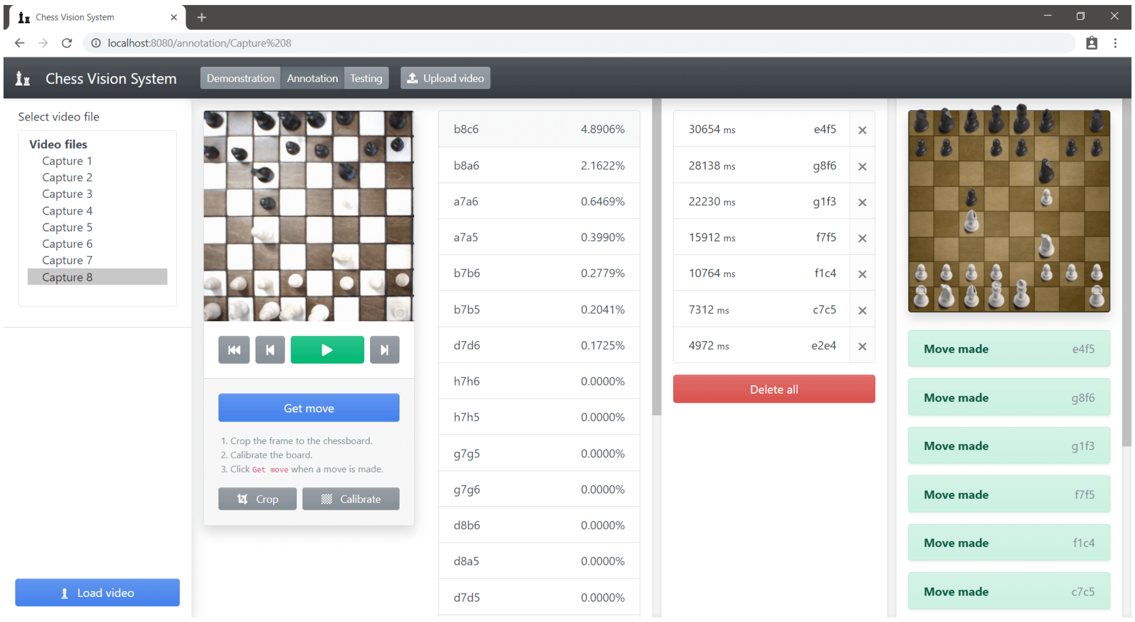
Video cropped

Figure 10: Screenshot of the demonstration page.

### 5.2.2 Annotation Page

The annotation page allows users to easily annotated videos of chess games. As shown in Figure 11, the annotation page retains the video playback and outputted state components as the demonstration page. However, it introduces two annotation components in the central pane; when a move is made, all possible moves are displayed along with the system's estimated probability of each move having occurred. When the user clicks the move which has been made, the move is recorded in the annotation file for the respective video. The list of annotated moves is also shown on the page, allowing the user to delete an annotation if required, for example, if a mistake has been made.

[16]https://chessboardjs.com



1. Crop the frame to the chessboard.
2. Calibrate the board.
3. Click `Get move` when a move is made.

| | |
|---|---|
| b8c6 | 4.8906% |
| b8a6 | 2.1622% |
| a7a6 | 0.6469% |
| a7a5 | 0.3990% |
| b7b6 | 0.2779% |
| b7b5 | 0.2041% |
| d7d6 | 0.1725% |
| h7h6 | 0.0000% |
| h7h5 | 0.0000% |
| g7g5 | 0.0000% |
| g7g6 | 0.0000% |
| d8b6 | 0.0000% |
| d8a5 | 0.0000% |
| d7d5 | 0.0000% |

| | | |
|---|---|---|
| 30654 ms | e4f5 | × |
| 28138 ms | g8f6 | × |
| 22230 ms | g1f3 | × |
| 15912 ms | f7f5 | × |
| 10764 ms | f1c4 | × |
| 7312 ms | c7c5 | × |
| 4972 ms | e2e4 | × |

| | |
|---|---|
| Move made | e4f5 |
| Move made | g8f6 |
| Move made | g1f3 |
| Move made | f7f5 |
| Move made | f1c4 |
| Move made | c7c5 |

Figure 11: Screenshot of the annotation page.

### 5.2.3 Testing Page

The testing page allows the user to test the computer vision system on multiple files in parallel, with their chosen configuration. The configuration pane initially displays the default values of the system's configuration: these values can be changed in order to tune the system, in order to find the optimum solution. When the desired configuration has been set up, clicking 'Run tests' will send a request for each selected file to be run asynchronously, and will update the percentage success of each video when they have completed.

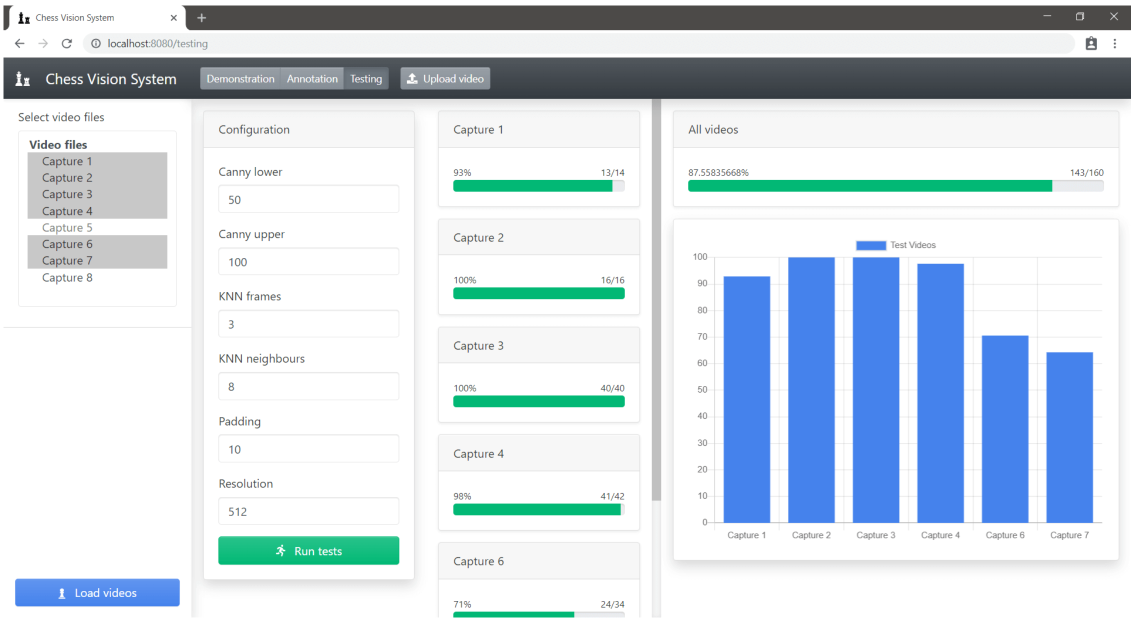
## 5.3 API Endpoints

The application programming interface allows the front-end of the system to interact with the computer vision system; in this case, the front-end of the system is the web application built with Vue, however other applications could be easily built to act as a client of the system.

As the project grew, the number of endpoints available to the client also grew, to allow for greater control of video playback and new features. In particular, new endpoints were created for:

- Rewinding and forwarding the video playback, to speed up annotation.
- Calibrating the board, to rotate the white pieces to the bottom.
- Annotating the videos, to record the position of the chessboard and the moves played.
- Returning the default configuration of the system, for the testing page.

The endpoints of the final API can be seen in Table 2, with the newly added endpoints highlighted in blue.



Configuration

Canny lower: 50

Canny upper: 100

KNN frames: 3

KNN neighbours: 8

Padding: 10

Resolution: 512

| | | |
|---|---|---|
| Capture 1 | 93% | 13/14 |
| Capture 2 | 100% | 16/16 |
| Capture 3 | 100% | 40/40 |
| Capture 4 | 98% | 41/42 |
| Capture 6 | 71% | 24/34 |
| All videos | 87.55835668% | 143/160 |

Figure 12: Screenshot of the testing page.

| Resource | Method | Description |
|---|---|---|
| `/devices/` | GET | Return the available webcam devices |
| `/videos/` | GET | Return the uploaded videos |
| `/videos/` | POST | Upload a new video file |
| `/videos/:video/load` | POST | Load the video with the given configuration |
| `/videos/:video/pause` | POST | Pause or unpause the video |
| `/videos/:video/restart` | POST | Restart the video playback |
| `/videos/:video/backward` | POST | Rewind the video playback by 5 seconds |
| `/videos/:video/forward` | POST | Forward the video playback by 5 seconds |
| `/videos/:video/crop` | POST | Crop the video to the given coordinates |
| `/videos/:video/calibrate` | POST | Calibrate the board |
| `/videos/:video/move` | POST | Return the probability of moves just played |
| `/videos/:video/play` | POST | Play the given move |
| `/videos/:video/annotate` | POST | Update the video annotation |
| `/videos/:video/annotation` | GET | Return the annotation for the video |
| `/videos/:video/stream/:type` | GET | Return the video stream of the given type |
| `/videos/:video/frame/:type` | GET | Return a single video frame of the given type |
| `/config/` | GET | Return the default configuration values |

Table 2: Final endpoints of the API and their function, after implementation.

## 5.4 Evaluation of the System

Perhaps the most important part of this project, the computer vision system must be well-tested so as to ascertain the accuracy of the system and determine the successfulness of the project's aims and objectives.

### 5.4.1 Chess Game Annotation

In order to easily evaluate the system's performance over multiple video files at once, and to allow for custom configuration of the system, an annotation framework was built. At the heart of this is the `VideoAnnotation` class, which was created in addition to the original class hierarchy discussed in Section 4.2. The `VideoAnnotation` class encompasses three things:

- The position of the chessboard, denoted by an array of coordinates.
- The time at which the chessboard should be calibrated (to ensure the board is not obscured by, for example, a player's hand).
- The moves made in the game, stored as a dictionary which maps moves to the time they were made.

This class is mirrored by an `annotation.json` file, which is updated every time an annotation is made in the graphical user interface. An example annotation file can be seen in Appendix F.

### 5.4.2 Method of Evaluation

The system has been tested in three ways to determine the performance of the system and its robustness. The set of videos used for the evaluation of the system consisted of 385 moves of chess, over twelve games. Out of the twelve games, six games were played in normal conditions, the other six videos were filmed from varying angles in both the horizontal and vertical planes. As discussed in Section 3.5.1, for every move in a game the predicted move was compared with the annotated move; the performance measure is the percentage of correctly identified moves.

Firstly, tests were carried out to find the effect that each configurable parameter had upon the performance of the system. The three configurable parameters of the system were:

1. The upper threshold of the Canny edge detection algorithm
2. The amount of padding within each square of the chessboard
3. The processing resolution of the chessboard

Secondly, in order to determine the robustness of the system, the system was tested upon videos taken from a range of angles and lighting conditions. When investigating the effect of lighting condition upon the performance of the system, the collected data was not varied enough to test the system's limits. Therefore extra data was generated from the existing data by adjusting the brightness and contrast of the videos in VLC media player[17]. Although this is not a perfect representation of real-life lighting conditions, the generated data is sufficient to identify the trends of the system's performance. Observing the performance of the system across the range of angles and varied lighting will provide an insight into what factors the system relies upon.

Finally, the overall performance of the system was found. Using the results of the previous tests, the configurable parameters were tuned in order to produce the optimum solution. The tests were ran using 5-fold cross-validation to ensure the results did not suffer from bias or overfitting.

[17]https://www.videolan.org/vlc

# 6 Results and Discussion

This chapter presents and discusses the results of the testing of the computer vision system, with the aim of assessing its performance and robustness. Firstly, the effect of the system's parameters upon the performance of the system is found: these results inform the final tuning of the system in order to improve its performance. Secondly, the effect of the environment upon the system is used to indicate the system's robustness. Lastly, the overall performance of the system is evaluated, using a K-fold cross-validation method.

The standard error was calculated for each set of results by the equation $\sqrt{\frac{p(1-p)}{n}}$, where $p$ is defined by $\frac{\text{Correctly identified moves [\%]}}{100}$ and $n$ is the total number of moves in that test case. The standard error ensures that the size of the test set is considered when evaluating the results of the tests. Although the total test set consists of 384 moves, the data is from only 12 chess games and the data within each game is expected to be highly correlated.

## 6.1 The Effect of the System's Parameters Upon Its Performance

The three configurable parameters of the system were tuned to find their optimum values, which corresponded to the highest percentage of correctly identified moves. After a parameter had been tuned, the optimum parameter value was retained for the subsequent tests. Initially, the square padding was set to 10% and the resolution was set to 512px.

The upper threshold of the Canny edge detection algorithm was varied between 0 and 255 in intervals of 20. The effect of the threshold upon the accuracy of the system can be seen in Figure 13. A low upper threshold can be seen to negatively impact the performance of the system. The optimum threshold is around 80 – after this, higher thresholds decrease the performance.

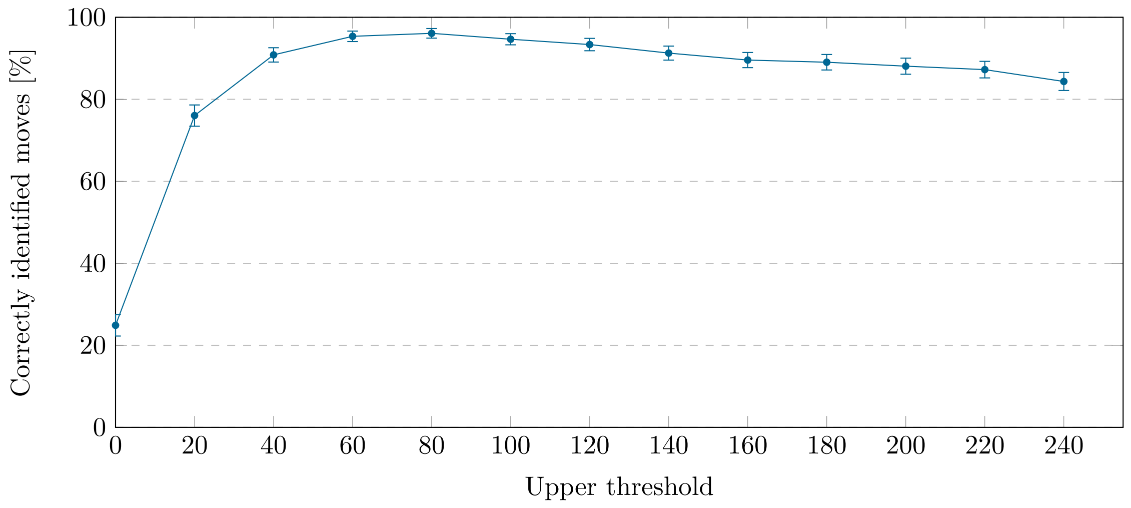


Figure 13: The effect of the upper Canny threshold upon the performance of the system.

Each square of the chessboard can be padded to avoid misclassification due to the occlusion of pieces in adjacent cells. The effect of varying square padding can be seen in Figure 14. The optimum padding seems to lay between 10% and 20% of the width of the cell. Less padding than this will not prevent the occlusion of pieces; any more padding than this will reduce the information available for edge detection and object classification.

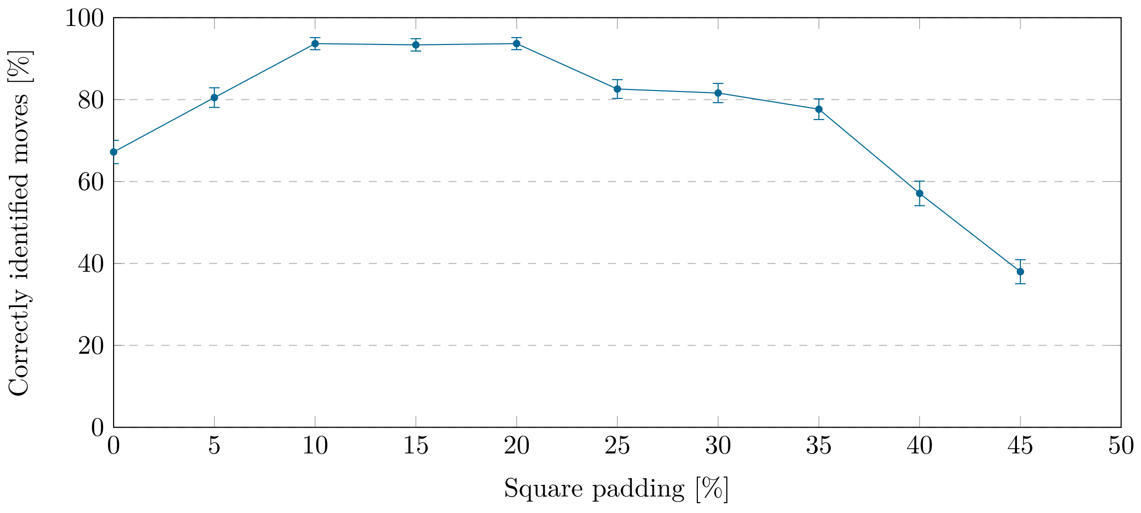


Figure 14: The effect of square padding upon the performance of the system.

The last parameter to be varied is the resolution of the chessboard. This resolution was varied in powers of two from 16px all the way up to 512px; multiples of 8 were chosen to ensure each square of the chessboard would consist of an integer number of pixels in width. The effect of varying chessboard resolution can be seen in Figure 15. Increasing the resolution incurs a dramatic increase in accuracy for lower resolutions, however as resolution increases past 128px this is less pronounced. The optimum resolution was found to be 384px.

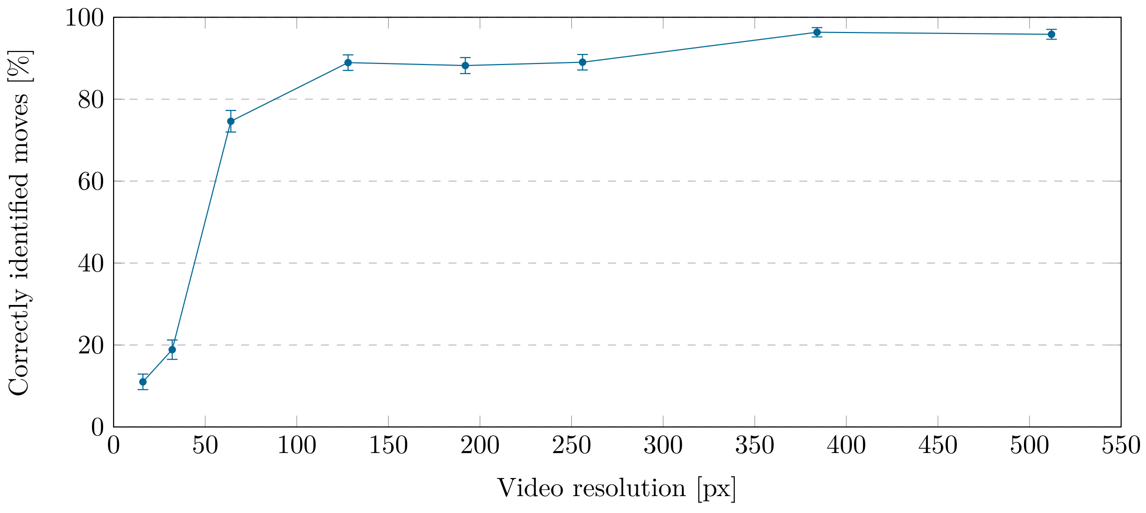


Figure 15: The effect of video resolution upon the performance of the system.

After tuning these three parameters one by one, the same tests were run a second time using the tuned parameters. The same trends were prevalent as before, implying that the three parameters are independent of each other, therefore no further tuning was needed. In summary, the best settings for the parameters of the system are: the Canny edge detection upper threshold set to 80; the padding of each square set to between 10% and 20%; and the resolution of the chessboard at 384px. These optimised parameters were used when exploring environmental factors in the subsequent section and were used when determining the overall performance of the system, in Section 6.3.

## 6.2 The Effect of the Environment Upon The System's Performance

Four factors were tested in order to assess the robustness of the system. This included the angle of the webcam in both the horizontal and vertical plane, and the brightness and contrast of the chessboard due to the lighting conditions. Figure 16 shows how the angle of the webcam, in both the horizontal and vertical planes, affects the performance of the system. The vertical angle of the webcam can be clearly seen to negatively affect the performance of the system at greater angles from the vertical, i.e. closer to the playing surface. Despite the system only being analysed on 14 moves at 70° from the vertical, with consideration of the standard error of the point, the performance of the system clearly decreased.

The effect of the webcam angle in the horizontal plane was tested between 0° and 40°, with 0° being parallel to the ranks of the chessboard. As seen in Figure 16, the horizontal angle of the webcam has a negligible impact on the performance of the system; at all angles, the performance of the system remained above 90%. The point at 40° shows a slight decrease in performance, however, given the standard error this can be assumed to be a result of the small sample size.

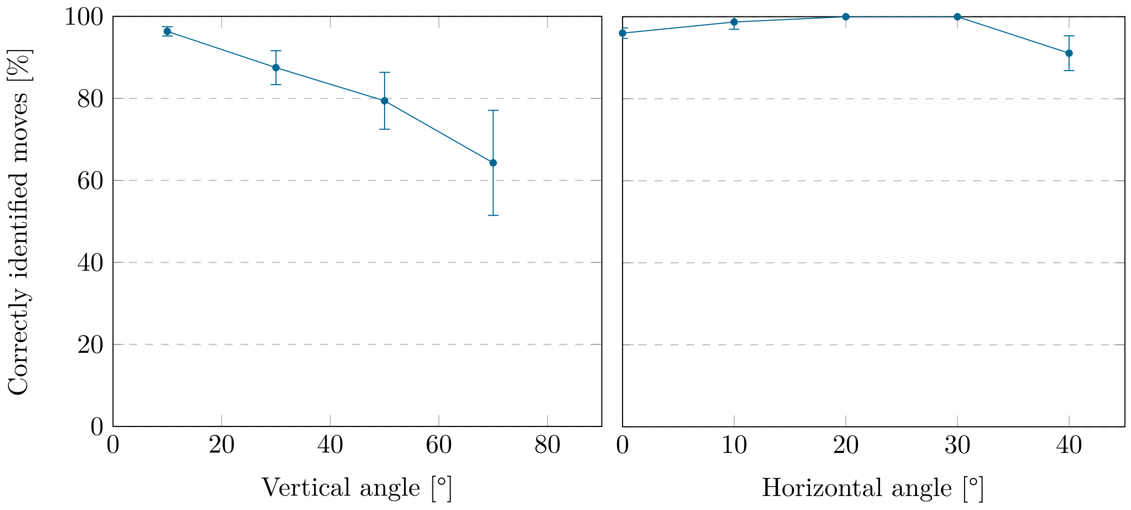


Figure 16: The effect of webcam angle upon the performance of the system.

The negative impact of the vertical angle of the webcam on the performance of the system can be expected, considering the occlusion of the chess pieces at greater angles from the vertical. Figure 17 shows the chessboard at the four angles tested in the vertical plane. When the webcam is placed close to the vertical, there is no occlusion between pieces so the system performs at its best. When the webcam is placed further from the vertical, the chess pieces start to overlap the square behind them which causes the moves to be misclassified; the square padding helps to minimise this but fails to prevent the problem at extreme angles.

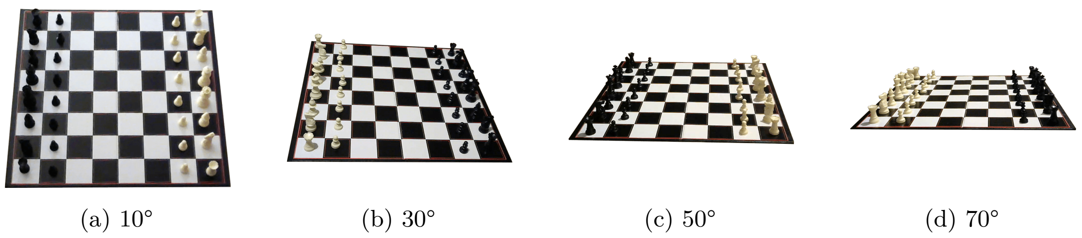


Figure 17: Screenshots of chess games at a range of vertical angles.

When determining the system's dependence on certain light conditions, brightness and contrast were measured. As previously discussed, in order to test the system on a sufficiently varied data set (with respect to lighting conditions), some data was generated; on the subsequent graphs, the limits of the real data is denoted by the dashed red lines.

In order to measure the brightness of the image, the average intensity across the chessboard was calculated. This prevents the measurements being affected by objects next to the board, which are not expected to affect the system's performance. As shown in Figure 18, it was found that the performance of the system remained roughly the same when varying the brightness of the board. However, in extremely dimly lit conditions, the performance of the system decreased as many of the black pieces were no longer visible.

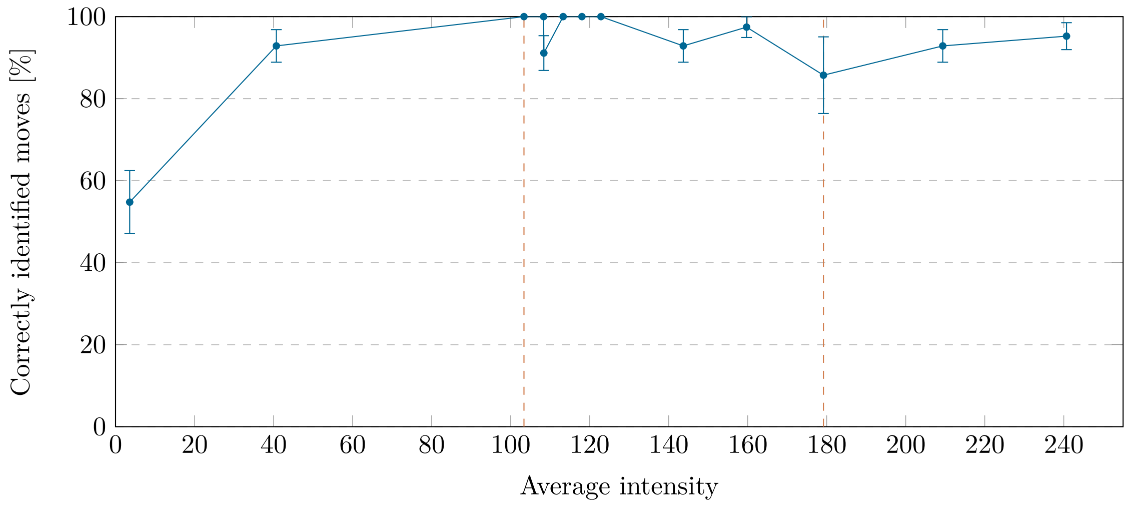


Figure 18: The effect of brightness upon the performance of the system.

The contrast of the image was measured by the average variance across each square of the chessboard. The effect of contrast upon the performance of the system can be seen in Figure 19; similar to the effect of brightness, the system's performance is only impacted at the extremes.

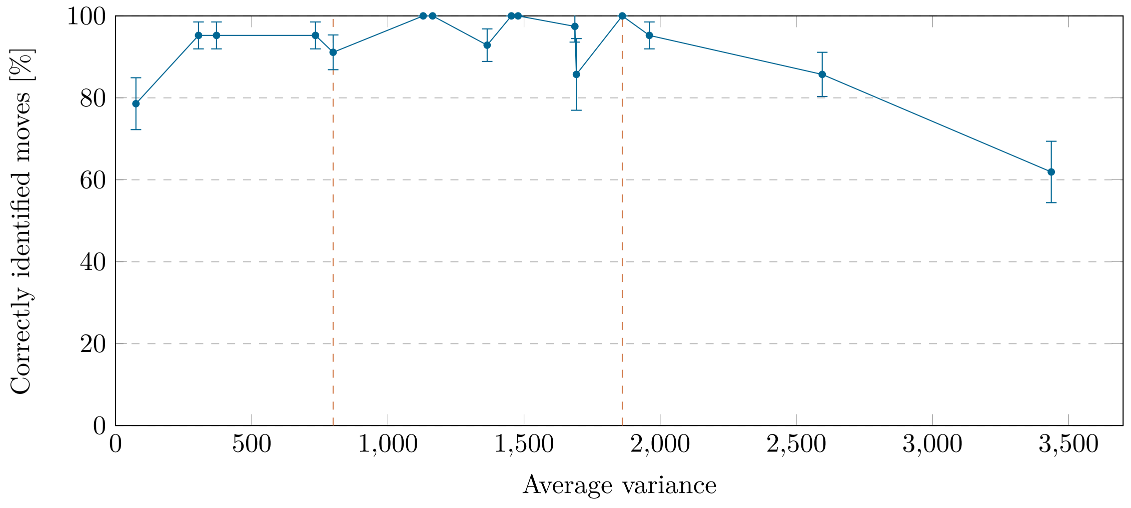


Figure 19: The effect of contrast upon the performance of the system.

## 6.3 Overall Performance of the System

The overall performance of the system was found by using 5-fold cross-validation upon the set of videos which constituted 'reasonable' conditions. As discussed in Section **??**, the only factor which consistently had a negative impact on the system's performance was the vertical angle of the webcam, so these videos were discounted. The remaining set of videos was split into 5 sets.

Often when using K-fold cross-validation it is recommended to shuffle the data set before splitting into groups, to avoid the bias of each set. However, in this case, all of the moves within each video are highly correlated: if these moves were shuffled, the training data would be very similar to some of the data in the testing set. This would give a false impression of the performance of the system, so it was decided to retain all moves in each video together in a set. The experimental results can be seen in Table 3.

| Set | Number of moves correct | Total number of moves | Correctly identified moves [%] |
|---|---|---|---|
| 1 | 41 | 45 | $91.1 \pm 4.2$ |
| 2 | 51 | 56 | $89.2 \pm 4.1$ |
| 3 | 57 | 57 | $100 \pm 0$ |
| 4 | 61 | 62 | $98.7 \pm 1.4$ |
| 5 | 53 | 53 | $100 \pm 0$ |
| Total | 263 | 273 | $96.3 \pm 1.2$ |

Table 3: Results of 5-fold testing on the overall performance of the system.

It has been shown that the average performance of the system is 96.3%, meaning that it should be able to correctly identify 96.3% of moves within a chess game. Although the standard error of the overall performance has been calculated to be $\sqrt{\frac{0.963\times(1-0.963)}{273}} = 1.2\%$, this does not seem to reflect the actual error seen across the 5 sets. This is again due to the highly correlated nature of the chess games. In order to find the true performance of the system on *any* given chess game, it may be prudent to instead take the number of samples as 5 i.e. the number of sets. Following through with this calculation, the standard error is calculated as $\sqrt{\frac{0.963\times(1-0.963)}{5}} = 8.4\%$, which is much more representative of the error shown across the collected data set. Therefore, the overall performance of the system may be better presented as $96.3\% \pm 8.4\%$.

# 7 Conclusions

The aim of this project was to create a computer vision system able to detect moves in a chess game, with no presumptions made regarding lighting conditions, the chessboard, or the webcam being used. This project has outlined the design of a system which successfully meets these criteria.

The problem of building a computer vision system which was as robust as it was performant posed many challenges for the author. The literature review into existing computer vision systems for chess provided valuable insight into relevant techniques used for change detection and classification. In particular, the use of edge detection processed frames, proposed by Sokic and Ahic-Djokic, greatly improved the robustness of the system to changes in shadows and lighting conditions. Using this technique, coupled with the K-nearest neighbours algorithm, led to a system which correctly identified 96.3% of moves within the collected set of chess games and was robust to a range of lighting conditions and the horizontal angle of the webcam.

The development of the web application early in the process greatly helped with the development of the back-end of the system, as changes to the move detection algorithm were immediately visible in the graphical user interface. The use of Vue and Bootstrap enabled a rapid development process, which would not have been possible if using solely HTML and CSS to create the web application.

Possibly the greatest challenge of this project was the collection of enough data in order to evaluate the system's performance accurately. The process of ethics approval to contact the university's chess society was a worthy investment for the amount of real-life data collected. If this project was taken further, it may be worth generating test data using 3D modelling software, as discussed in Section 3.4.2. If the simulated games were a good representation of real-life data, the system could be tested much more thoroughly; this would allow for further exploration into novel algorithms to detect the moves in a chess game.

In conclusion, this project has been challenging but very rewarding. The realisation of the system shows promising performance, whilst ensuring that minimal constraints are placed upon it. There is scope to see the project be taken further. With a larger set of chess videos to train the system upon, new algorithms and techniques could continue to improve the performance and robustness of the system, with the aim of creating a fully autonomous chess-playing robot.

## References

Bootstrap Team (2019) ‘The most popular HTML, CSS, and JS library in the world.’, *Bootstrap*. Available at: https://getbootstrap.com/ (Accessed: 20 April 2019).

Bouwmans, T. and Garcia-Garcia, B. (2019) ‘Background Subtraction in Real Applications: Challenges, Current Models and Future Directions’. Available at: http://arxiv.org/abs/1901.03577 (Accessed: 13 April 2019).

‘Chess’ (2017) *Encyclopædia Britannica*. Encyclopædia Britannica, inc. Available at: https://www.britannica.com/topic/chess (Accessed: 11 April 2019).

Computer Chess Rating Lists (2019) ‘CCRL 40/4’, *CCRL*. Available at: http://ccrl.chessdom.com/ccrl/404/index.html (Accessed: 11 April 2019).

Computer Vision LAB (2013) ‘Change Detection Algorithms’, *Computer Vision LAB, DISI, University of Bologna*. Available at: http://vision.deis.unibo.it/research/78-cvlab/85-change-detection (Accessed: 13 April 2019).

Django Software Foundation (2019) ‘The Web framework for perfectionists with deadlines’, *Django*. Available at: https://www.djangoproject.com/ (Accessed: 17 April 2019).

Farooque, M. A. and Rohankar, J. S. (2013) ‘Survey on Various Noises and Techniques for Denoising the Color Image’, *International Journal of Application or Innovation in Engineering & Management (IJAIEM)*, 2(11), pp. 217–221.

Fiekas, N. (2019) ‘Python-chess’. Available at: https://github.com/niklasf/python-chess (Accessed: 11 April 2019).

Grinberg, M. (2014) ‘Video Streaming with Flask’, *miguelgrinberg.com*. Available at: https://blog.miguelgrinberg.com/post/video-streaming-with-flask (Accessed: 20 April 2019).

Huang, T. S. (1996) ‘Computer Vision: Evolution and Promise’. Available at: http://cds.cern.ch/record/400313/files/p21.pdf.

IBM (2012) ‘Deep Blue’, *IBM100*. Available at: http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/ (Accessed: 13 April 2019).

Jain, R., Kasturi, R. and Schunck, B. G. (1995) *Machine Vision*. McGraw-Hill New York.

Kahlen, S.-M. (2004) ‘Description of the Universal Chess Interface (UCI)’, *UCI protocol*. Available at: http://wbec-ridderkerk.nl/html/UCIProtocol.html (Accessed: 13 April 2019).

Koray, C. and Sümer, E. (2016) ‘A Computer Vision System for Chess Game Tracking’.

Levitt, G. M. (2000) *The Turk, Chess Automation*. McFarland & Company, Inc. Publishers.

MathWorks (2019) ‘Object Recognition’, *MathWorks MATLAB*. Available at: https://uk.mathworks.com/solutions/deep-learning/object-recognition.html (Accessed: 13 April 2019).

Pallets Team (2010) ‘Foreword’, *Flask Documentation*. Available at: http://flask.pocoo.org/docs/1.0/foreword/#what-does-micro-mean (Accessed: 17 April 2019).

Piccardi, M. (2004) ‘Background Subtraction Techniques: A Review’, in *2004 IEEE International Conference on Systems, Man and Cybernetics*. IEEE, pp. 3099–3104.

Python Software Foundation (2019) ‘Thread-Based Parallelism’, *Python Documentation*. Available at: https://docs.python.org/3/library/threading.html (Accessed: 17 April 2019).

Schiller, E. (2003) *The Official Rules of Chess*. Cardoza Pub.

Shannon, C. E. (1950) ‘Programming a Computer for Playing Chess’, *Philosophical Magazine*, 41(314).

Sokic, E. and Ahic-Djokic, M. (2008) ‘Simple Computer Vision System for Chess Playing Robot Manipulator as a Project-based Learning Example’, in *2008 IEEE International Symposium on Signal Processing and Information Technology*, pp. 75–79. doi: 10.1109/ISSPIT.2008.4775676.

The United States Chess Federation (2015) ‘New to Chess? FAQ’, *The United States Chess Federation*. Available at: http://www.uschess.org/content/view/7328/28/ (Accessed: 12 April 2019).

Torborg, S. (2019) ‘Python-packaging’. Available at: https://github.com/storborg/python-packaging (Accessed: 16 April 2019).

Vue Team (2019) ‘Introduction’, *Vue.js*. Available at: https://vuejs.org/v2/guide/ (Accessed: 20 April 2019).

Wong, K. D. (2009) ‘Canny Edge Detection Auto Thresholding’, *www.kerrywong.com*. Available at: http://www.kerrywong.com/2009/05/07/canny-edge-detection-auto-thresholding/ (Accessed: 29 April 2019).

Yang, M.-H. (2009) ‘Object Recognition’, in *Encyclopedia of Database Systems*.

# Appendices

## A Ethics – Consent Form

### Researchers

Student: Gregory Ives (gjives1@sheffield.ac.uk)

Student: Lawrence Burvill (ljburvill1@sheffield.ac.uk)

Supervisor: Jon Barker (j.p.barker@sheffield.ac.uk)

### Declaration

1. I confirm that I have read and understood the information sheet dated March 16, 2019 explaining the above research project and have had the opportunity to ask questions about the project.
2. I understand that my participation is voluntary and that I am free to withdraw at any time without giving any reason and without there being any negative consequences.
3. I understand that the recorded video(s) of me playing chess are to be kept private, solely used for this research project and that my identity is to be kept confidential. I understand that my name will not be linked with the research materials, and I will not be identified or identifiable in the report or reports that result from the research.
4. I understand that images from the data may appear in the publicly available project report.
5. I agree to take part in the above research project.

| | | |
|---|---|---|
| Name of participant<br>(or legal representative) | Signature | Date |
| Name of researcher | Signature | Date |

To be signed in the presence of the participant.

### Copies

Once all parties have signed this, the participant should receive a copy of the signed and dated participant consent form, the letter/pre-written script/information sheet and any other written information sheet provided to the participants. A copy of the signed and dated consent form should be placed within the project's main record (e.g. a site file), which must be kept in a secure location.

**Please address any queries to: Gregory Ives or Lawrence Burvill**

# B Ethics – Information Sheet

## Researchers

Student: Gregory Ives (gjives1@sheffield.ac.uk)

Student: Lawrence Burvill (ljburvill1@sheffield.ac.uk)

Supervisor: Jon Barker (j.p.barker@sheffield.ac.uk)

## Invitation

You are being invited to take part in a research project. Before you decide whether to participate or not, it is important for you to understand why the research is being done and what it will involve. Please take time to read the following information carefully and discuss it with others if you wish. Ask us if there is anything that is not clear or if you would like more information.

## Aim

The purpose of this project is to create a system which can detect moves being played in a chess game by way of a webcam or alternate video recording device, in a range of environments. After detecting a move, the system will then be able to pass the result to a number of applications, including a chess-playing robot. In order to train the system, it will require many hours of video of chess games; the data will be used solely for this research project.

## Your Task

You will be asked to play a number of games of chess against an opponent. A webcam will be placed above the board and directed at the board. It will capture your hands as you play chess moves, and will not capture any identifying information such as your face. You may play as many games as you wish, and you are not required to finish every game you play.

## Confidentiality

No personal data will be recorded for this project. The only data collected will be the video of you playing chess, with the webcam directed at the board from above: therefore, it will capture your hands as you play a move and no identifying information. Audio will be muted when recording the video, so no audio data will be captured or stored. Your identity will remain entirely confidential. Your name will not be linked to the research materials, and you will not be identified or identifiable in any reports that result from the research.

[Continued overleaf]

## How the Video Will Be Used

The video data will be used for training and evaluating the system. Still images from the video may appear in the project report for illustrative purposes.

## Consent

It is up to you to decide whether or not to take part. If you do decide to take part you will be given this information sheet to keep, and will be asked to sign a consent form. You can still withdraw at any time without there being any negative consequences. You do not have to give a reason.

Date: March 16, 2019

**Please address any queries to: Gregory Ives or Lawrence Burvill**

# C Ethics – Application Form

## Application 025573

### Section A: Applicant details

| | |
|---|---|
| Date application started: | Sat 16 March 2019 at 17:23 |
| First name: | Greg |
| Last name: | Ives |
| Email: | gjives1@sheffield.ac.uk |
| Programme name: | Computer Science |
| Module name: | COM3610 |
| Last updated: | 18/03/2019 |
| Department: | Computer Science |
| Applying as: | Undergraduate / Postgraduate taught |
| Research project title: | Computer Vision System for a Chess-Playing Robot |
| Has your research project undergone academic review, in accordance with the appropriate process? | Yes |
| Similar applications: | *- not entered -* |

### Section B: Basic information

#### Supervisor

| Name | Email |
|---|---|
| Jon Barker | j.p.barker@sheffield.ac.uk |

#### Proposed project duration

Start date (of data collection):
Tue 19 March 2019

Anticipated end date (of project)
Wed 1 May 2019

#### 3: Project code (where applicable)

Project code
*- not entered -*

### Suitability

Takes place outside UK?
No

Involves NHS?
No

Human-interventional study?
No

ESRC funded?
No

Likely to lead to publication in a peer-reviewed journal?
No

Led by another UK institution?
No

Involves human tissue?
No

Clinical trial?
No

Social care research?
No

Involves adults who lack the capacity to consent?
No

Involves research on groups that are on the Home Office list of 'Proscribed terrorist groups or organisations?
No

### Indicators of risk

Involves potentially vulnerable participants?
No
Involves potentially highly sensitive topics?
No

## Section C: Summary of research

### 1. Aims & Objectives

The purpose of this project is to create a system which can detect moves being played in a chess game by way of a webcam or alternate video recording device, in a range of environments. After detecting a move, the system will then be able to pass the result to a number of applications, including a chess-playing robot. In order to train the system, it will require many hours of video of chess games; the data will be used solely for this research project.

### 2. Methodology

Subjects will be asked to play a number of games of chess against an opponent. A webcam will be placed above the board and directed at the board. It will capture the players hands as they play chess moves, and will not capture any identifying information such as their faces. They may play as many games as they wish, and they are not required to finish every game they play. Audio will not be captured, i.e. the microphone will be muted.

### 3. Personal Safety

Have you completed your departmental risk assessment procedures, if appropriate?

Not applicable

Raises personal safety issues?

No

The subjects will just be playing chess in a typical environment, albeit with a webcam pointed at the board; this poses no risk to their safety.

## Section D: About the participants

### 1. Potential Participants

We are looking to recruit anyone who is able to play chess.

### 2. Recruiting Potential Participants

We will primarily contact the Chess Society at the University of Sheffield, however we may contact others who can play chess and are willing to join.

### 2.1. Advertising methods

Will the study be advertised using the volunteer lists for staff or students maintained by CiCS? No

*- not entered -*

### 3. Consent

Will informed consent be obtained from the participants? (i.e. the proposed process) Yes

Participants will be provided with a clearly written Information Sheet (see attachment) that will be made available to them as soon as they are recruited. They will be given the opportunity to ask questions and care will be taken to make sure that they understand what we will be doing with their data. They will be asked to sign the consent form (see attachment) before the experiment commences.

### 4. Payment

Will financial/in kind payments be offered to participants? No

### 5. Potential Harm to Participants

What is the potential for physical and/or psychological harm/distress to the participants?

There is a possibility that whilst playing chess in front of a webcam, a participant may become stressed.

How will this be managed to ensure appropriate protection and well-being of the participants?

It will be made clear to all participants that they may stop at any point, with no negative ramifications.

## Section E: About the data

### 1. Data Processing

Will you be processing (i.e. collecting, recording, storing, or otherwise using) personal data as part of this project? (Personal data is any information relating to an identified or identifiable living person).
No

Please outline how your data will be managed and stored securely, in line with good practice and relevant funder requirements

The video data will only be distributed to the project researchers, for training and evaluation purposes. Still images from the video may appear in the publicly available research reports.

## Section F: Supporting documentation

### Information & Consent

Participant information sheets relevant to project?
Yes

Document 1058075 (Version 1) All versions

Consent forms relevant to project?
Yes

Document 1058076 (Version 1) All versions

Additional Documentation

External Documentation

*- not entered -*

## Section G: Declaration

Signed by:
Gregory Ives
Date signed:
Sat 16 March 2019 at 17:51

## Offical notes

*- not entered -*

# D Ethics – Approval Letter

The University Of Sheffield.

Downloaded: 15/04/2019
Approved: 18/03/2019

Greg Ives
Registration number: 160152746
Computer Science
Programme: Computer Science

Dear Greg

**PROJECT TITLE:** Computer Vision System for a Chess-Playing Robot
**APPLICATION:** Reference Number 025573

On behalf of the University ethics reviewers who reviewed your project, I am pleased to inform you that on 18/03/2019 the above-named project was **approved** on ethics grounds, on the basis that you will adhere to the following documentation that you submitted for ethics review:

- University research ethics application form 025573 (dated 16/03/2019).
- Participant information sheet 1058075 version 1 (16/03/2019).
- Participant consent form 1058076 version 1 (16/03/2019).

If during the course of the project you need to deviate significantly from the above-approved documentation please inform me since written approval will be required.

Yours sincerely

Lucy Moffatt
Ethics Administrator
Computer Science

## E Live Webcam Using OpenCV and Flask

```python
from flask import Flask, Response
import cv2

app = Flask(__name__)

# Generator of video frames
def gen(video):
    while True:
        # Read frame from video
        success, image = video.read()
        ret, jpeg = cv2.imencode('.jpg', image)
        frame = jpeg.tobytes()
        # Yield new frame of video
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')

@app.route('/')
def video_stream():
    # OpenCV webcam capture
    video = cv2.VideoCapture(0)
    # Multipart response
    return Response(gen(video),
        mimetype='multipart/x-mixed-replace; boundary=frame')

app.run()
```

## F Example Annotation File

```json
{
  "crop": [
    [
      0.27023498694516973,
      0.12309254070001995
    ],
    [
      0.7193211488250653,
      0.13933855172409884
    ],
    [
      0.7297650130548303,
      0.9330722331862399
    ],
    [
      0.2506527415143603,
      0.9214679395976121
    ]
  ],
  "calibrate": 0.0,
  "moves": {
    "4913.985560807907": "e2e4",
    "7019.979372582724": "c7c5",
    "10178.97009024495": "f1c4",
    "15619.45410399656": "f7f5",
    "21586.436570691876": "g1f3",
    "27670.418693596905": "g8f6",
    "32701.403910614525": "e4f5",
    "38375.88723678556": "b8c6",
    "53585.84254404813": "a2a4",
    "58616.827761065746": "g7g5",
    "67859.800601633": "d2d4",
    "79033.2677696605": "c6d4",
    "100444.20485603782": "c1e3",
    "114074.66480446927": "d8a5"
  }
}
```