A computer vision approach to playing the board game English Draughts through NAO robot technology

Russell Wilson

A dissertation submitted to

The school of Computing Sciences of the University of East Anglia

in partial fulfilment of the requirements for the degree of

MASTER OF SCIENCE

AUGUST, 2021

SUPERVISOR(S), MARKERS/CHECKER AND ORGANISER

The undersigned hereby certify that the markers have independently marked the dissertation entitled **"A computer vision approach to playing the board game English Draughts through NAO robot technology"** by **Russell Wilson**, and the external examiner has checked the marking, in accordance with the marking criteria and the requirements for the degree of **Master of Science**

Supervisor: _____

Dr. R. Lapeer

Markers: _____

Marker 1: Dr. Lapeer

_____

Marker 2: Dr. Day

External Examiner: _____

Checker/Moderator

Moderator: _____

Dr. Wenjia Wang

DISSERTATION INFORMATION AND STATEMENT

Dissertation Submission Date: **August, 2021**

Student: **Russell Wilson**

Title: **A computer vision approach to playing the board game English Draughts through NAO robot technology**

School: **Computing Sciences**

Course: **Computing Science**

Degree: **MSc**

Duration: **2020 - 2021**

Organiser: **Wenjia Wang**

STATEMENT: Unless otherwise noted or referenced in the text, the work described in this dissertation is, to the best of my knowledge and belief, my own work. It has not been submitted, either in whole or in part for any degree at this or any other academic or professional institution. Subject to confidentiality restriction if stated, permission is herewith granted to the University of East Anglia to circulate and to have copied for non-commercial purposes, at its discretion, the above title upon the request of individuals or institutions.

Signature of Student

Abstract

NAO robot technology represents the future for robotics and applied computer vision. NAO robots provide an introduction to concepts of robotics and computer vision for learned and developing programmers alike. Currently, the available online and physical resources that demonstrate the potential and capabilities of NAO robotics remain sparse, with visibility of NAO projects using computer vision and written in Python nearly non-existent. This document outlines a computer vision model built in Python2 that utilises the OpenCV library to identify and classify counters in the game of English Draughts through an image captured by a V4 NAO robot. The proposed solution, uniquely, does not operate with the support of machine learning concepts but can classify counters in the game with an average accuracy of 83%. However, this paper finds with further optimisation of processes documented within this body of work, the average accuracy obtained could be significantly higher.

ACKNOWLEDGEMENTS

I would first like to thank my supervisor Dr Rudy Lapeer, whose experience, wisdom and humour played a pivotal role throughout the lifespan of my dissertation.

I would also like to acknowledge Dr Wenjia Wang for his support throughout my time at university and thank my friends, Roberto and Matthew for their valuable time spent proofreading.

Finally, I would like to offer a special thanks to my partner Amber, mother Stephanie, father Stephen and little sister Ariane, who always play an indispensable role in all aspects of my work, life and beyond.

Russell Wilson

Norwich, UK

Contents

List of Figures

1 List of Abbreviations

API Application programming interface

CPU Central processing unit

SVM Support vector machine

USP Unique selling point

RGB Red, green and blue colour channels

BGR Blue, green and red colour channels

SBR SoftBank Robotics

CNN Convolutional neural network

KPI Key performance indicator

2 Introduction

2.1 Background

The necessity of this work is in the response to an omission of work focused on implementing NAO robotics with Python, using computer vision concepts. NAO is a robot that has many applications outside of computer vision, such as natural language processing operations for example. In cases of documented work with computer vision through NAO, whilst there may be names of image processing algorithms, the exact implementation process is not defined which makes it extremely difficult to replicate. NAO is an extremely capable form of robotics and accessible to beginner programmers, while offering much flexibility in the types of applications it can be used for. Providing information on more ways to utilise NAO, serves to promote innovation within the computer vision community and spark the creativeness of young developers that will fuel academic growth through the application of NAO.

2.2 Research objective

In accordance with the background, the objective of this work is to test a combination of NAO technology and computer vision in the process of playing the game, English Draughts. Specifically, this work seeks to test if the application of NAO with computer vision can produce an accurate, computationally cheap and timely system that is capable of identifying counters in the game draughts and selecting a legal optimal move.

Research hypothesis

In measuring the success of the work documented and the results obtained from system implementation a hypothesis statement is established. This can be given as, ”It is a viable application to utilise NAO robotics and computer vision, unsupported by machine learning, in detection and placement of the counters in the game draughts”.

2.3 Document structure

This document contains seven key sections, summarised as follows.

Literature Review

Firstly, a background to computer vision is established to ensure the reader understands where this work is positioned with relation to computer vision and its application. Next, a high level overview of robotics is discussed to ensure the benefits of utilising a NAO robot against alternatives is understood. Finally, relevant academic work is discussed to provide context to the chosen algorithms suggested in this work.

Methodology

This work assumes the reader may have minimal or little experience with computer vision, therefore this section is required to equip the reader with a basic foundation of knowledge regarding all of the algorithms used within this project. With a foundation of knowledge, the reader can critically review the approach taken in this work.

Design

This section starts by outlining the rules for English Draughts to remove confusion that stems from alternative known variations of the game. The project methodology applied in implementing the programmed system is then discussed and a quick overview of the process with reference to the evaluation criteria is also discussed.

Implementation

The stages of implementation of the proposed programmed solution are then defined in detail, from start to end with challenges of implementation also discussed.

Results

The results of the implementation are then tested against the evaluation criteria, established in the Design stage.

Evaluation

Following the collection and delineation of implementation results, the provided results

will be critically evaluated with reference to academic literature to analyse the implementation process objectively. Following this stage, a conclusion will provide a summarisation of work produced in this document and provide discussion ideas for future work before concluding.

3 Literature review

3.1 Computer vision viewed holistically

When defining a computer vision project it is first important to examine the scope of computer vision. IBM (2021) defines computer vision as a field, ”that enables computers and systems to derive meaningful information from digital images, videos and other visual inputs — and take actions or make recommendations based on that information”. Inline with this definition, computer vision can be seen as a product of two core elements, digital image generation and data retrieval resulting in new data insight.

A Camera Processing Pipeline is the title given to the preliminary steps taken that govern the creation of a digital image, the various stages include rendering, colour correction, demosaicing, denoising and encoding.

There is a bountiful amount of work produced by academics which focuses on reviewing and improving the camera processing pipeline. From the plethora of work there are many suggestions on how to reduce computational requirements to greatly improve image capture speed and image quality. Jayasuriya et al. (2017) considers the classic camera processing pipeline, see Appendix A.1, and outlines an adjustable sensory solution, which given a set of required image parameters, will produce the required image by removing camera pipeline stages from the camera pipeline that have high computational cost and contribute a small amount to the final constructed image, this solution is estimated to save 75% of the average energy cost in producing images from modern camera technology. Producing any work suggesting improvements to the image capture process would be highly contested and would not be likely to produce any new perspective to the field of computer vision.

When we consider image processing techniques there are sufficient omissions in literature to pursue studies highlighting new techniques to improve image processing or introduce new constructs in image processing. Xiaofeng & Bo (2012) innovated on the classical method of object detection with contour detection by introducing support vector machines (SVM) for increased accuracy in contour detection. Contour detection is applied in

many applications of object detection and image segmentation, including the research documented in this dissertation. There is also a possibility to find new insight by critiquing and comparing algorithms and methods in image processing. Matuska et al. (2012) explored two leading computer vision libraries (OpenCV and Matlab) to compare CPU performance when applying computer vision algorithms through each library. The final consideration for needed academic work lies in the exploratory application of computer vision image processing, specifically applying adaptions of known computer vision algorithms to solve new problems not yet challenged by other academics. Academic work focused on exploratory applications could be considered integral to global economic growth, increasing the world's understanding of computer vision capabilities while also producing many social benefits, the focus of this study. Grand View Research (2020), valued the 2019 global computer vision market at $10.6 billion and indicated the market is likely to grow at 7.6% annually from 2020 to 2027. Khemasuwan et al. (2020) outlined the potential of computer vision in identifying cases of COVID-19 through pulmonary examination with the support of machine learning and deep learning concepts. Xia et al. (2018) highlighted the necessity of applying computer vision concepts in tracking biological markers in aquatic life to indicate dangerous levels of water toxicity. Whilst computer vision is not a new concept, it remains evident that the field continues to grow and through the integration of other forms of technology, such as NAO robotics, there exist too many unrealised global rewards from the product of computer vision to have the field ignored.

3.2 NAO technology

NAO robot technology is a product of SoftBank Robotics (SBR), formally known as the French company, Aldebaran. Robots produced by SBR have a humanoid look, which is very apparent with the models NAO and Pepper. The NAO robot, in addition to the humanoid look, supports a variety of motors and actuators that provide humanoid like movement and are both innovative and patented. All versions of the NAO robot from the version 3.2 to the version 5.0 come with integrated video cameras, contact sensors and a microphone. "Its exclusive actuation system based on brush DC motors and modularity of the robot limbs allows further evolution and aid maintainability. These are the innovative traits that distinguish NAO from its Japanese, American and other counterparts", Shamsuddin et al. (2011). In addition to NAO, SBR have produced Pepper, a humanoid robot designed for improved human interaction compared to the NAO robot. Pepper in contrast to NAO

delivers some different features that include improved facial tracking and a improved microphone array. However, Pepper has reduced points of movement due to a wheel based movement system, where NAO offers a kinematic walking system which is much like human movement. The Pepper robot was designed after the release of the NAO robot and like NAO has undergone many upgrades and improvements, however Pepper production has been discontinued as many businesses terminated their hire contracts for Pepper owing to an insufficient business user case. The initial USP of Pepper was an autonomous machine able to successfully read human emotions and respond, very useful in a sales environment such as retail, however Metz (2021), confirms that even with the assistance of IBM, an industry leading business in the market of machine learning and AI concepts, the Pepper robot was not able to successfully read human emotions and was re-purposed by SBR as a general assistance robot. Though SBR offer the Whiz Cobot cleaning robot, due to the specialised function of the robot, while exemplifying the values of computer vision in solving common and laborious problems such as cleaning, research focused on integrating new or different computer vision concepts with the Whiz Cobot is unlikely to benefit further research, and so should be limited. Further studies based on the application of Pepper in computer vision have the potential to produce much merit for the computer vision field, however as Pepper is to be discontinued currently. The most promising SBR solution that can be applied with computer vision concepts and produced new studies is the NAO Robot.

The robotics industry much like the computer vision industry is growing exponentially each year. Statista (2019) expects the global industrial and non-industrial robotics market to be worth nearly $210 billion by the year 2025, please also see Appendix A.2 for year by year growth. The ability for robotics to automate the boring, physically taxing and necessary jobs that bring value to people and businesses is unrivaled. Both industrialised and non-industrialised applications of robots such as the Whiz Cobot, that applies programmed cleaning routines, utilise computer vision concepts to perform tasks. Karuppiah et al. (2018) applied combined computer vision image segmentation by colour for object detection with a wheelchair mounted robotic arm for automatic recognition and collection of household items, assisting someone with limited mobility. Yeotikar et al. (2016), combined a computer vision template matching algorithm with a robotic arm in the implementation of dental implants for patients with great success. Shamsuddin et al. (2011) highlighted that NAO technology is ground breaking in terms of current global robotic capabilities, as the demand for computer vision applications and robotics increases globally, further research into ap-

plications of NAO and computer vision are justified and essential for academic growth.

3.3 Related work

There exists an abundance of work that focuses on introducing computer vision concepts to board games, specifically the game chess which, likely sparked by the popularity of IBM's Deep blue and its prestigious success over Garry Kasparov in 1994, has become the gold standard for applications of computer vision and board games. Most recently Wölflein & Arandjelović (2021) attempted to recognise and store the positions of game pieces in the game of chess through computer vision programming assisted by a convolution neural network (CNN), which is an applied machine learning concept. Interestingly, the resulting solution was able to detect pieces on the board with an accuracy of 93.86% where no errors exist. Additionally, the speed of computation was also very impressive, with the proposed system capable of real-time detection at speeds of two frames per second. Many publications explore the possibility of chess with AI and computer vision assisted by machine learning concepts owing to the exciting new prospective of combining all these techniques. Recently, work published on the use of robotics in the games of chequers or draughts, which are fundamentally similar, has been exclusively orientated on integrating new and improved AI techniques aimed at making robotic gameplay as intelligent as possible. Meng (2019) explored the idea of implementing an AI system using a bespoke Monte Carlo tree search algorithm to best an AI applying an Alpha-Beta pruning algorithm. It has been noticed, that there exist few publications focused on the use of computer vision in draughts, specifically the techniques and methods to identify the type of piece and position of the piece relative to the draught's board location. Kopets et al. (2020) chose to adopt magnetised counters and modified a chequers board with hall effect sensors placed on each game square, ensuring the magnetise chequers could be detected at all positions on the board. Lewis & Bailey (2004) published a brief snapshot of a project that implemented computer vision to enable a robotic arm to detect a chequers gameboard, select an optimal move and move the chequer autonomously. Unfortunately, the publication does not feature information on the relevant computer vision steps taken to successfully implement computer vision into the project and leaves the work hard to replicate. There then exists an omission of work that is purely focused on applying and documenting the methods of computer vision applied in draughts, which could help students and professionals alike to further understand the capabilities of computer vision.

Interestingly, it is not unique to see robotics applied in computer vision or to the game chequers as can be seen in Lewis & Bailey (2004). However, there is lesser publicised work that looks at applied computer vision with NAO, but NAO remains a versatile form of engineering, extremely capable in applications of computer vision. Magallán-Ramirez et al. (2021) programmed NAO robots to successfully communicate with each other and provide instructions on how to complete a maze. The processed involved one virtual robot attempting to complete a 3d replicate of the maze, storing the shortest path from maze start to exit and then communicated this to a physical NAO robot which then used the information to complete the physical version of the maze. The project employed Canny edge detection and projective transformations successfully with NAO to establish boundaries of the mazes, techniques that are adopted as part of this documented project. Unlike most other projects involving NAO robots, the project opted to program the robots in Python2 using OpenCV and scikit-learn. Conversely, Szemenyei & Estivill-Castro (2018) introduced a real time object detection system programmed in C++, that used deep learning CNNs in real-time to detect objects in a game of football played by NAO robots. It is evident to see the speed of C++, the compiler language, when applied in a computer vision applications thanks to Szemenyei & Estivill-Castro (2018), However, it is Python, the interpreted language, that is a much more accessible language to those that are new to programming. As this work is encouraged to be a learning exercise, it is necessary to program in Python to ensure that it is easily replicated by many others. At the time of writing this document, it has not been possible to find any work that applied computer vision through NAO technology to play the game draughts. This is interesting as NAO has been designed to look and feel human, unlike a lot of other robots that show no expressions and are built purely for industrial applications, the NAO robot has the potential to be found in a domestic setting where some of its likely applications may include playing games such as draughts. It is therefore necessary to investigate draughts played with NAO to spark further creativity among academics and inspire further work focused on applications of NAO in computer vision. This work will stand to further entrench NAO as a lightweight yet effective robotic solution for computer visions problems.

4 Methodology

This section is focused on providing context to the computer vision algorithms employed within this project. By reviewing these algorithms and their structure with reference to

alternative algorithms, the reader should appreciate the logic behind the direction of this project while gaining a solid foundations of knowledge to review the implementation critically.

4.1 Computer vision image processing concepts

When considering a computer vision project there is an abundance of computer vision algorithms and computer vision libraries that can be adopted. OpenCV and Matlab are two of the leading libraries for computer vision and accommodate all types of computer vision projects, alternative libraries tend to focus more specifically on machine learning and deep learning in computer vision, which is not necessary within this project. OpenCV is an open source library while Matlab is a licensed library, and both allow the design of computer vision solutions in languages such as Python or C++ however, Matuska et al. (2012) noted that OpenCV is less computationally expensive for computer vision applications than Matlab and is a lighter solution for computer vision. Owing to OpenCV computational lightness and the ability to easily replicate the work in this document it was decided to adopt OpenCV for the project documented in this work.

4.1.1 Image resizing

Image resizing is a core function in computer vision. An image may be enlarged to increase visibility of key image features or alternatively, an image may be shrunk to reduce the computational requirements of computer vision algorithms. By reducing the number of pixels within the image the computational requirements are reduced. The core OpenCV resizing algorithms are:

- **Bilinear interpolation**
- **Nearest neighbor interpolation**
- **Bicubic interpolation**

Bilinear interpolation

When increasing or decreasing the size of an image, Bilinear interpolation can be used to interpolate the function of a missing pixel's value in the resized image, in a cost effective

manner.

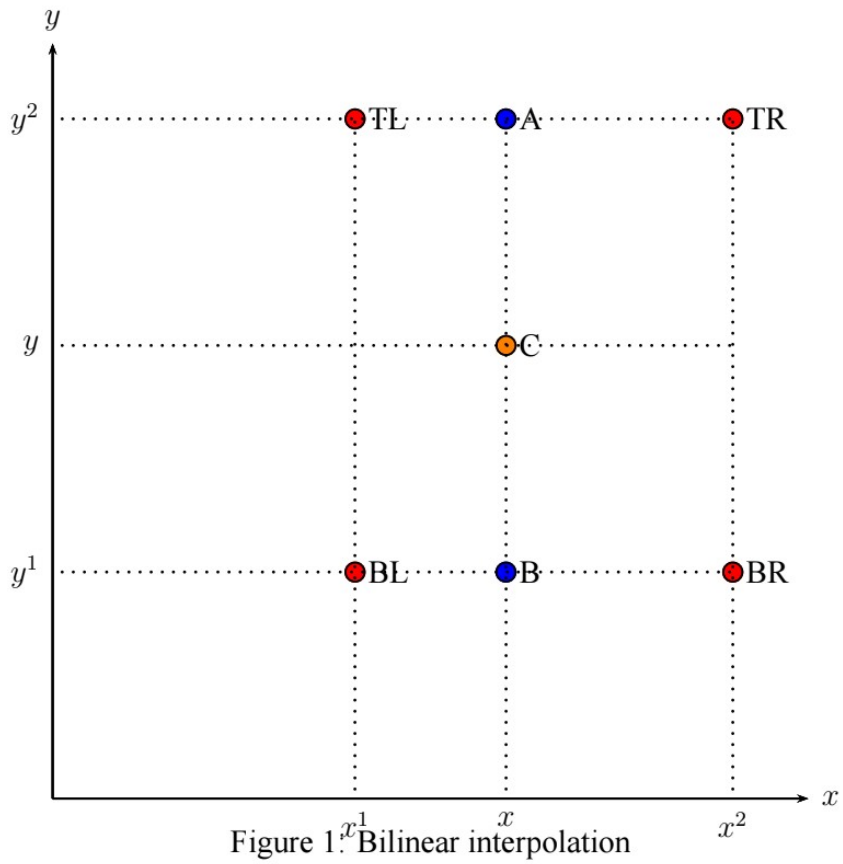


Figure 1: Bilinear interpolation

First an image is increased or decreased by a scaling factor, reflecting the change in the ratio of size with respect to the initial image. For example, reviewing Figure 1 it is possible to see an enlargement operation where the functions of the known pixel RGB values from the original image are given as

$$TL, TR, BL, BR$$

However after the resizing the image there are now unknown pixel RGB values which can be seen as:

$$A, B, C$$

the unknown values can be calculated by applying Bilinear interpolation.
Firstly to find the value of A and B respectively, linear interpolation is applied. The value for A can be given as a function of TL and TR, where:

$$f(A) \approx \frac{x_2 - x}{x_2 - x_1} f(TL) + \frac{x - x_1}{x_2 - x_1} f(TR) \quad \text{where} \quad A = (x, y_2)$$

and

$$f(B) \approx \frac{x_2 - x}{x_2 - x_1} f(BL) + \frac{x - x_1}{x_2 - x_1} f(BR) \quad \text{where} \quad B = (x, y_1)$$

And the by applying Bilinear interpolation it is possible to solve for C , where:

$$f(C) \approx \frac{y_2 - y}{y_2 - y_1} f(A) + \frac{y - y_1}{y_2 - y_1} f(B)$$

Solving by Bilinear interpolation is a straight forward mathematical operation and can be less computationally expensive than other resizing operations whilst be extremely efficient.

Nearest Neighbour Interpolation

The nearest neighbour method is a relatively simplistic algorithm, favoured due to the speed of computations. The nearest neighbour interpolates the unknown functions of pixels by taking the ratio of pixels in the original image to the new image and multiplying each pixel by the ratio. Consider a 3x3 matrix of pixels that represents an image, given by the matrix below:

$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}$$

To enlarge the matrix to a 6x6 image using nearest neighbour interpolation the ratio of pixel from the old to the new image must be calculated and are given below:

$$Ratio_{row} = \frac{3}{6} \quad \text{and} \quad Ratio_{column} = \frac{3}{6}$$

With the known ratios the new image matrix can be calculated according to the $Ratio_{row}$ and $Ratio_{column}$ values, see below:

$$NewImage = \begin{bmatrix} 1 & 0 & 2 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 2 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 2 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 & 2 & 2 & 3 & 3 \\ 1 & 1 & 2 & 2 & 3 & 3 \\ 1 & 1 & 2 & 2 & 3 & 3 \\ 1 & 1 & 2 & 2 & 3 & 3 \\ 1 & 1 & 2 & 2 & 3 & 3 \\ 1 & 1 & 2 & 2 & 3 & 3 \end{bmatrix}$$

While the nearest neighbour may be the simplest interpolation algorithm, an image that has been enlarged will not have weighted RGB values for each pixel. Instead the image will have a distinct range of pixels with specific RGB values which will cause the image to look pixelated when enlarging the image with any significant scaling. A Bilinear interpolation

will ensure edges are smoother and less pixelated as RGB values are calculated according to a weighted pixel distance.

Bicubic Interpolation

Bicubic Interpolation works much like Bilinear interpolation, however the function of a pixel is determined through a 4x4 grid of pixels surrounding the unknown pixel value instead of a 2x2 pixel grid. The missing pixel value is then calculated in the same manner as Bilinear interpolation. considering figure 1, the enlargement operation under Bicubic interpolation now becomes figure 2.

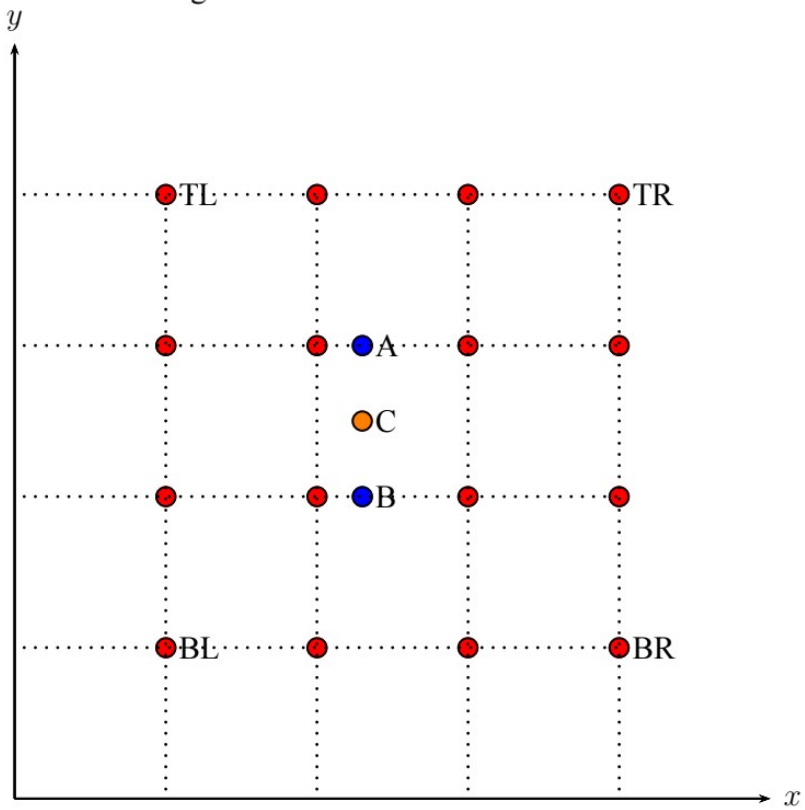


Figure 2: Bicubic Interpolation

With the additional sub-pixel range to calculate, the pixel C will be more reflective of the weighted RGB values shared by the 16 pixel neighbourhood in comparison to the standalone Bilinear interpolation. This will better assist cases where there are different light contrasts across an imagine and will likely give a smoother image with less bias towards a single pixel RGB value.

The Bilinear interpolation methods should be sufficient when appling image resizing to this project. Han (2013) noted that there was a 25% average increase in the processing times of resizing operations using Bicubic instead of Bilinear interpolation. The camera used in this project has a resolution of 1.22mp and is not likely to benefit greatly based on resolution

from Bicubic interpolation additionally, with limited processing power due to hardware limitations, Bilinear interpolation resizing is optimal for this project.

4.1.2 Morphological transformations

Morphological operations can be applied through the OpenCV library to remove image noise. Salt and pepper noise, specifically, represents a problem in image processing operations with a camera that has fewer sensors and reduced hardware, such as the camera in the NAO robot.

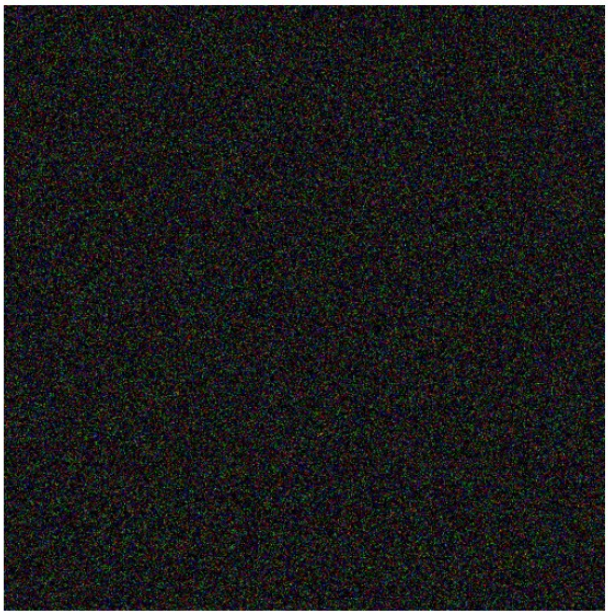


Figure 3: Salt and pepper noise

An example of salt and pepper noise can be seen in figure 3. Salt and pepper noise results from variations in image luminosity and shadow levels and can really hinder object detection by colour, owing to the differences in pixel RGB values that do not occur on the object but are created when the image of the object is processed. A combination of dilation and erosion operations can be adopted through OpenCV's opening algorithm to address salt and pepper noise, this is a processing step taken successfully by Paterson & Aldabbagh (2021) to improve gesture detection of a robotic arm.

Erosion

In an erosion operation the image to be eroded is adjusted by a structuring element, also known as the kernel. The structuring element can take any shape, however common opera-

tions take a 3x3 square matrix as the kernel, this can be seen in figure 4. Image convolution operations, are operations that adjust an original image according to a kernel matrix into a new image, such operations include sharpening, blurring and dilation operations. The general expression of a convolution operation can be given as:

$$g(x,y) = w * f(x,y) = \sum_{dx--a}^{\alpha} \sum_{dy--b}^{\beta} w(dx,dy) f(x+dx, y+dy)$$

where $g(x,y)$ represents the new image, $f(x,y)$ represents the old image and w represents the kernel.

1	1	1
1	1	1
1	1	1

Figure 4: Kernel Matrix

An image can be divided into a foreground and background, in an erosion operation the foreground is reduced and replaced by the background. When considering a binary black and white image, the erosion operation can be seen as follows. The kernel is first passed over an image, if the centre of the kernel is placed on a pixel with a background value, the pixels in the image surrounding the location of centre pixel will be adjusted according to the size of the kernel neighbourhood, the adjusted pixels will be changed to the background RGB range which is zero in this case. If the centre of the kernel is placed onto a foreground pixel, which has a value of one, no change takes place. This operation can be seen below.

$$originalimage \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} newimage \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

This operation is extremely effective when there is a single isolated foreground pixel surrounded by background pixels, like in the case of salt and pepper noise. By removing the

isolated pixel image processing operations become easier when applying object detection methods.

Dilation

Image dilation operations work like an erosion operation but in a different direction. Taking again a binary image, first a structuring element usually, a 3x3 matrix square kernel, is passed through an image. When the centre of the kernel is placed on a foreground pixel value, in this case a value of one, the pixels in the image surrounding the location of the centre pixel will be adjusted according to the size of the kernel neighbourhood, the adjusted pixels will be changed to the foreground RGB range which is one in this case. If the centre of the kernel is placed onto a background pixel, which is has a value of zero, no change takes place. This operation can be seen below.

$$original\,image \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad new\,image \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Dilation operations can improve object detection activities by improving edge definitions. A combination of an erosion operations and dilation operations, known as an opening operation, can be applied from the OpenCV library to first remove noise which isolates and removes single pixel noise. Once the isolated pixel is removed, the image can be dilated to the original foreground ratio. This an essential step in image processing, that was applied effectively by Paterson & Aldabbagh (2021).

4.1.3 Greyscale conversion

A greyscale conversation is a simple image processing operation which takes a coloured image represented by a 3D array and converts it into a greyscaled image, which is represented by a 2D array, by transforming each pixel of a coloured image into a greyscale value which lies between a binary value. Greyscale images are less computationally ex-

pensive to processes, owing to having less depth. This makes converting coloured images into greyscale images an essential requirement for most computer vision processing. A greyscale conversion can be performed through averaging the current colour, in the case of an RGB colour range the greyscale value can be given as:

$$F(x, y) = \frac{R + G + B}{3}$$

A greyscale conversion can also be performed by converting a pixel value by a chosen weighting, where W is given as an assigned weighting and the general formula can be given by:

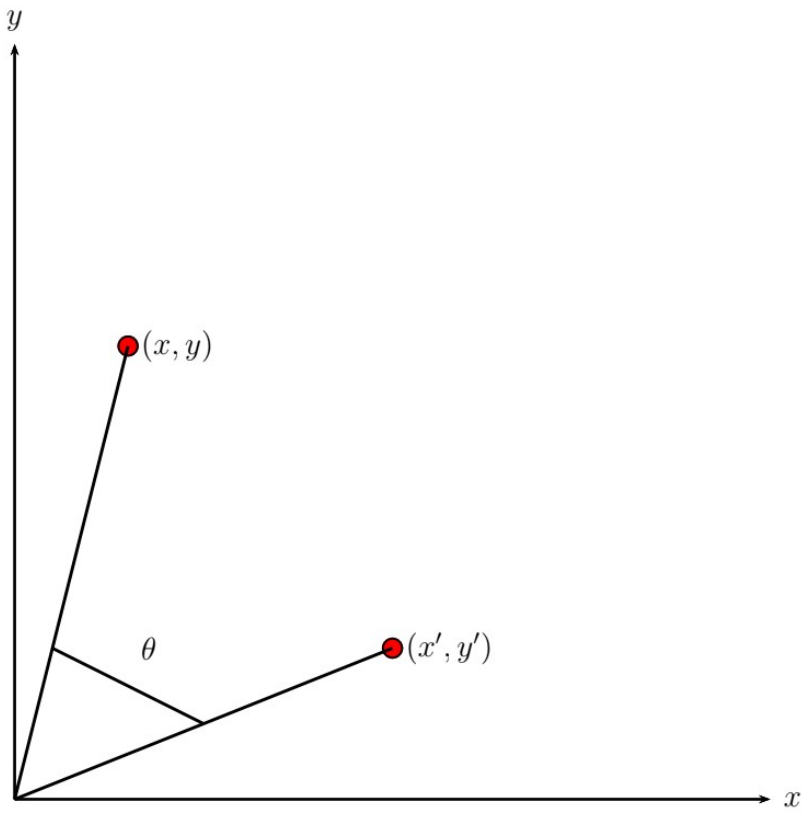
$$F(x, y) = (W_1 * R) + (W_2 * G) + (W_3 * B)$$

and

$$W_1 + W_2 + W_3 = 1$$

4.1.4 2D transformations

Digital image transformation can be an essential step in successful image processing projects. Wölflein & Arandjelović (2021) applied protective transformations to create a top down view of a chess gameboard. Common 2D image transformation operations included rotating or scaling an image, in these transformations operation the pixel location, given by (x, y) is passed through a matrix M_R that will generate the new pixel location (x', y').Considering a rotation operation, that can be seen below



Where the new pixel translation can be given by

$$\begin{aligned} x' &= xcos(\theta) - ysin(\theta) \\ y' &= xsin(\theta) + ycos(\theta) \end{aligned} \tag{1}$$

the transformation matrix can then be given as

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Projective transformations and affine transformations are slightly more complex than rotation operations or scaling operations. These operations involve transforming images on a 3D plane and can be performed with homogeneous coordinates, where (x, y) becomes (x, y, w) and w represents the pixel depth. Project transformations unlike affine transformation change the perspective of the image, such as in the case of Wölflein & Arandjelović (2021). The homogeneous transformation matrix can be given as

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = M_R \begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} h_{11} & h_12 & h_13 \\ h_21 & h_22 & h_23 \\ h_31 & h_32 & h_33 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

4.1.5 Gaussian blurring

Blurring operations in computer vision are effective in removing noise from images, specifically salt and pepper noise. A blurring operation is a convolution operation that distorts the original images by removing definition and blurring the pixels. In understanding how blurring can reduce image noise, a basic blurring or filtering operation with a greyscale image is recommended. Through averaging with a 3x3 kernel it is possible to reduce the image noise, as can be seen below.

$$original, image \begin{bmatrix} 90 & 20 & 55 \\ 10 & 255 & 90 \\ 100 & 120 & 140 \end{bmatrix} \rightarrow New\, image \begin{bmatrix} 90 & 20 & 55 \\ 10 & 98 & 90 \\ 100 & 120 & 140 \end{bmatrix}$$

In the example above the centre pixel in the original image can be given as a and the centre pixel in the new image can be given by b where b can be calculated as the average from the kernel neighbourhood in this case.

$$b = \frac{a}{k} \rightarrow 98 = \frac{255}{9}$$

It is possible to see that the high pixel value of 255, which is the maximum value of white, has been adjusted to 98 in accordance to the average of the pixel neighbourhood and therefore the noise, which can be seen as the imbalance in the neighbour, has been corrected. A Gaussian blur assumes a Gaussian or normal distribution of pixel's RGB values from the centre of a kernel to the kernel boundary. The kernel values are weighted, with each deviation away from the centre pixel the weight is reduced, with closer pixels having greater weighting on the centre pixel colour. Gaussian blurring is more effective at preserving definition in a blurred image than basic blurring methods such as averaging. As Gaussian filtering empathises the weighting of pixels closer to the centre of the kernel, when the kernel is placed on an edge, the immediate pixels external to the centre are high determinants of the pixel RGB range, ensuring edges are more defined than if blurring through a non weighted neighbour average. The general equation of Gaussian distribution can be given as:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} \exp^{-\frac{x^2+y^2}{2\sigma^2}}$$

Where σ gives the standard deviation of the distribution and (x, y) provides the location

of the pixel. Once a Gaussian kernel is established as can be seen in the example below, it is possible to see that the highest weightings are placed on the centre pixel of the kernel and the pixels closest to the kernel centre. To produce the blurred image a convolution operations passes the Gaussian kernel through the original image and generates a new image, an example Gaussian kernel is produced below.

$$\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix} \rightarrow Gaussian\,Kernel \begin{bmatrix} 0.004 & 0.015 & 0.03 & 0.015 & 0.004 \\ 0.015 & 0.06 & 0.10 & 0.06 & 0.015 \\ 0.03 & 0.10 & 0.15 & 0.10 & 0.03 \\ 0.015 & 0.06 & 0.10 & 0.06 & 0.015 \\ 0.004 & 0.015 & 0.026 & 0.015 & 0.004 \end{bmatrix}$$

Increasing the size of the Gaussian kernel for example to a 15x15 matrix greatly increases the convolution processing time and therefore the computation time, however Gaussian blurring is extremely effective at removing noise and is a necessary operation in object detection operations. Palekar et al. (2017) applied Gaussian blurring to remove noise in a real-time licence plate reader. Figure 5 below

Figure 5: From left, Non Gaussian blurred image, 5x5 Gaussian blur to one σ

4.1.6 Canny edge detection

The Canny edge detection algorithm, developed by John F. Canny, Canny (1986), is popular in image processing operations involving object detection. Canny edge detection can be applied to a greyscale image, separating and highlighting all possible edges within an image,

as can be seen in figure 6.

Figure 6: Canny edge detection operation

In applying Canny edge detection there are five essential steps

1. Gaussian Filtering
2. Gradient calculation
3. Non-maximum suppression
4. Double threshold
5. Edge Tracking by Hysteresis

To calculate the gradient of a pixel the Sobel or Prewitt filters can be used. The Sobel filter was developed by Irwin Sobel, Sobel & Feldman (1968). In applying the Sobel filter a two kernel convolution takes place on a greyscale image, with the two kernels emphasising vertical and horizontal edges separately. The magnitude of the directional gradients are then calculated to provide a new pixel value, which will provide a white level, highlighting a line. The Prewitt filter, created by Judith M. S. Prewitt, Prewitt (1970), is nearly identical to the Sobel filter except that it does not weigh pixel values based on distance from the kernel centre. This can reduce the intensity of an edge and make it less visible. Ahmed (2018), noted that the Prewitt filter convolution was performed quicker than the Sobel filter convolution, however the Sobel filter was able to better identify diagonal edges while being slower but not significantly. The Sobel filters in the horizontal and vertical direction can be seen below where G_x gives the horizontal filter and G_y the vertical filter

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

where the magnitude of the gradient can be given as G_m

$$G_m = \sqrt{G_x^2 + G_y^2}$$

Additionally, as the gradient is now known the orientation of the pixel can be calculated according to the formula:

$$\theta = arctan(G_x, G_y)$$

Applying the Sobel filter produces an image output similar to figure 7

Figure 7: Sobel filtered image

In figure 7 it is possible to see that edges have been identified within the images by applying a horizontal and vertical filter. However, there is still a lot of noise within the filtered image and due to differences in gradients there are thicker and thinner whites lines in the image. The variance in line thickness has the potential to affect object detection operations such as contour detection through line imbalance, so is then a necessary step to adjust for this. Non-maximum suppression can be used to remove lines with weaker gradient intensities, leaving only pixels with stronger white value intensities and therefore stronger edges. Consider an edge that is orientated horizontal and consists of three values. To calculate the

centre pixel value, each pixel value directly after and before the centre is considered. If the centre pixel value intensity is greater that its neighbouring pixels in the same edge, the pixel value is preserved, if the value is smaller than either neighbouring pixel, the value is set as zero and removed from the edge. This technique can be demonstrated below.

$$\begin{bmatrix} 210 & 40 & 20 \end{bmatrix} \rightarrow \begin{bmatrix} 210 & 0 & 20 \end{bmatrix}$$

After applying the non-maximum filtering, the thickness of edges are reduced which makes the image clearer, this can be seen in figure 8, however there exists some broken edges that can be improved by applying a thresholding operation. A classic threshold operation involves taking a single threshold value and utilising this to produce a binary output. In a greyscale image, a threshold value of 200 will ensure all pixels with a value lower than 200 become 0 and all pixels with a value greater than 200 become 1. After non-maximum filtering, multi level thresholding is applied. Multi-level thresholding separates strong pixel intensities, weak pixel intensities and weaker pixel intensities, where strong pixels greatly contribute to an edge, weak pixels contribute in some form to an edge and weaker pixels can be considered unnecessary noise in edge detection. The threshold levels in multi-level thresholding are arbitrary but will feature a high and low level threshold, with all pixel values over the higher threshold equal to 1, all pixel between the higher and lower threshold will have a value equal to x where $0 < x < 1$ and all pixels below the lowest threshold will be equal to 0.

Figure 8: Non-maximum suppression

The Final step in Canny edge detection is to apply edge Tracking by hysteresis. Previous attempts to reduce image noise may cause edges to be broken and incomplete, edge tracking by hysteresis attempts to reconnect broken edges through image convolution where a kernel is passed over the image, the centre of kernel is compared to the kernel neighbourhood, if the neighbourhood contains at least 1 strong pixel intensity the centre pixel is adjusted to the maximum intensity. If the kernel neighbourhood contains no strong pixel intensities the centre pixel is unchanged. Once these two additional steps are preformed, the Canny edge detection is complete, this can be seen in figure 9

Figure 9: Canny edge image

4.1.7 Image contours

A contour can be defined as a curve that joins a collection of identical points by pixel intensity, such as all white pixels with the value 1. Edge detection differs from contour detection slightly, the purpose of edge detection is to identify edges visually and preform a threshold operation that binarizes the edges, not returning the position, shape or size of an edge. A contour operation exists to detect a curve based on the location of a collection of continuous identical pixels. After edge detection is applied contour detection will detect edges and provide the location, size, and hierarchical information of the edge. Contours are often used in object detection operations and can be useful when trying to determine a shape type, area, parameter or order. OpenCV provides access to a single contour detection algorithm, the Suzuki contour detection algorithm proposed by Suzuki et al. (1985), and while many algorithms such as Moore Boundary tracing and Square tracing exist, the project in

this document applies the OpenCV library exclusively for computer vision, therefore it is not necessary to expand on further work. Suzuki et al. (1985) proposed two algorithms for contour detection. These algorithms are image convolution operations that involve passing a kernel across an image to identify outer and internal boundaries of an image, additionally the order of each contour with respect to the contour's immediate parent contour is recorded, providing the contour's external boundary. Both the outer and inner boundary kernel can be seen below

$$Contour_{matrix} = \begin{bmatrix} k, j \end{bmatrix}$$

$$Outer\,boundary\,kernel\,(OB) \begin{bmatrix} 0 & 1 \end{bmatrix} \quad Inner\,boundary\,kernel\,(IB) \begin{bmatrix} 1 & 0 \end{bmatrix}$$

$$Contour = C_i$$

The first step in contour operation is to pass OB through an image from left to right, at this point the count of the external boundary EB is given as $EB = 1$ where the only know external boundaries are the sides of the full image and are therefore the largest external boundary. Where $OB_J \geq 1$, this is the position of the first outer boundary and the count of the external boundaries is now:

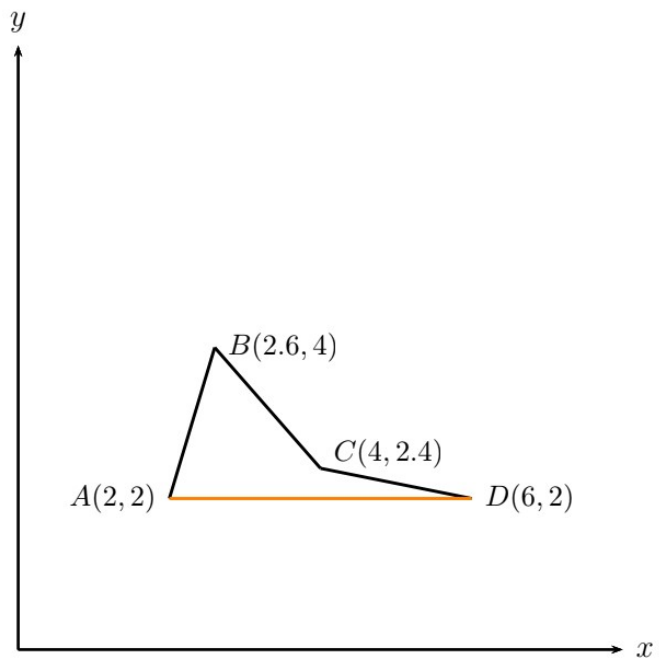
$$EB + 1 = EB \equiv 1 + 1 = 2$$

The OB kernel then continues right until $OB_J = 0$, all the preceding pixels to this point are identified as the second contour, where $C_2 = ENB$. Moving through the image at the next observed pixel (x, y) if $(x, y) \geq 1$ and $(x_{+1}, y) = 0$, also if (x,y) is within the width of C_2, (x, y) it is now a internal boundary (INB) and $(INB = -ENB{-}1) \equiv (INB = -3)$. This process continues until all internal contours are identified. If the observed (x,y) is equal to $C_2(x_{min}, y)$ or $C_2(x_{max}, y)$ it is included as a continuous point in the contour C_2.

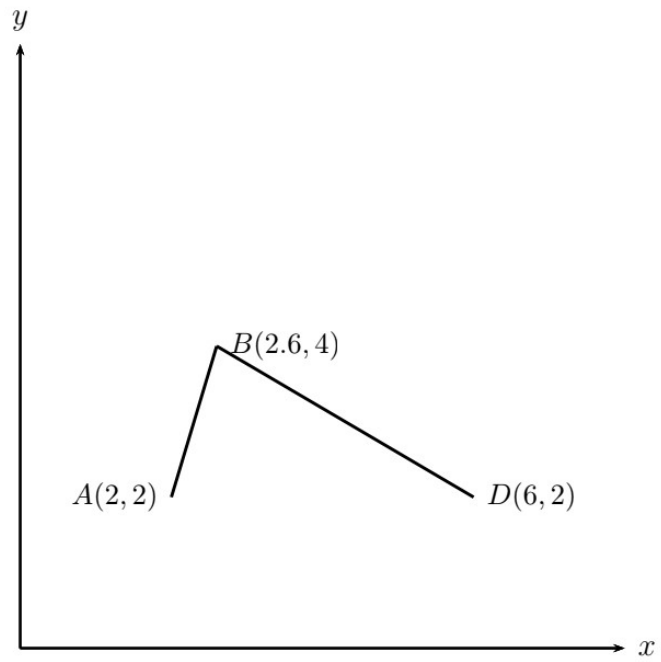
As previously discussed, understanding the geometry of a contour is extremely useful in certain applications. Poda & Qirici (2018) applied contour detection and identification in determination of a polygon shape. Additionally, contours are extremely adaptable to project requirements and the Ramer-Douglas-Peucker algorithm is a useful technique to improve object detection with contouring, by assisting in changing the characteristics of a closed polyline which results in different polygons.

Ramer-Douglas-Peucker algorithm

The algorithm suggested in 1973 by Douglas & Peucker (1973) can be applied to change the characteristic of a continuous polyline, such as a contour. The algorithm considers the design of the polyline a construct of many smaller polylines. The algorithm's object is to iterate over known points on the line and attempt to remove some points and their connection to the original polyline without breaking the line connection from the starting point to the finish. Removing points from the line changes the shape and characteristics of the polyline, applying this to a contour that is a polygon will change the polygon dimensions. The algorithm's degree of change is decided by an arbitrary value ϵ. Considering an example line below, where $\epsilon = 0.5$.



Firstly a temporary line joins the start and end point, given by line AD in the above example.Next the furthest point from line AD is identified which is B. As the distance between AD and B is greater than 0.5 the polyline is broken down into two separate polylines AB and BCD. There are no points between the line AB and therefore this cannot be deconstructed. Moving to the next line BCD there is 1 point C between BCD. The distance between point C and the line BD is less that ϵ and therefore the point can be ignored. The new polyline ABDcan be seen below.



4.1.8 Histogram equalization

Image contrast plays a pivotal role in object detection and identification. Contrast in an image can be defined as the variance between pixel colours which results from a variance in luminosity. In an image with poor contrast it can be difficult to distinguish features such as edges, as pixel values are grouped closely together and not distributed uniformly. Histogram equalisation exists to improve contrast in images by attempting to ensure the uniform distribution of pixel values along the dynamic greyscale range, from 0 to 255.

Figure 10: Original image (left) and global histogram equalised image (right)

OpenCV (2021)

It is clearly visible from figure 10 that balancing out the pixels along the dynamic range helps to improve the detection of both background and foreground objects. Histogram equalisation can broadly be divided into two methods:

1. Global histogram equalisation

2. Local histogram equalisation

Global histogram equalisation

Global histogram equalisation takes a histogram created of the whole image, with the frequency of pixel occurrence known, the known probability of pixels occurring across an image is utilised to create an image transformation criterion where pixel values are uniformly distributed across the full dynamic greyscale range from 0 to 255, which can be seen in figure 10. To apply global histogram equalisation it is first necessary to compute the image histogram. The number of histogram buckets range from o to L where $L = 256$ and each bucket is uniformly spaced as to capture every possible level of pixel intensity. Each pixel $f(x, y)$ is placed into a single bucket and the sum of any one bucket h_i can be given mathematically as:

$$h_i = \sum_{x=i}^{W} \sum_{y=i}^{H} \begin{cases} 255 \\ 0 \end{cases}$$

Where w is the image width, h is the image height and 0 to 255 is the maximum dynamic range of an image. Next the cumulative distribution function of the histogram must be calculated, given as:

$$CDF_i = \sum h_i$$

The pixel intensity transformation for any given pixel can then be determined as:

$$f(x, y) = \Big(\frac{CDF_i - CDF_{min}}{(WxH) - CDF_{min}} \Big) xL - 1$$

where CDF_{min} is the cumulative frequency of the first non zero pixel value. By applying this transformation to all pixels values in the original image the image is now transformed, as can be seen in figure 10.

Local histogram equalisation

One problem with global histogram equalisation is the presence of noise. Small concentrated areas of noise in an image will contribute significantly to the CDF of the final output image. Local histogram equalisation is a solution that breaks down an image into multiple smaller histograms, determining a pixel intensity transformation based on the isolated histogram and then merging the results into one image. The benefit of this technique is that it

isolates the small areas of concentrated noise and uniformly distributes the intensity values over a smaller range of pixels, reducing the effect of noise compared to global histogram equalisation.

Contrast limiting equalisation

Contrast limiting is an additional step that can be taken to improve effectiveness of local histogram equalisation. Contrast limiting very simply, sets a minimum and maximum value for image intensities before the CDF transformation is calculated. Aarthy & Sumathy (2014) noted that local contrast limiting histogram equalisation preformed extremely well compared to other histogram equalisation techniques, however it was noted that a considerable disadvantage of the technique is speed of computation, which can be a significant disadvantage in large image processing or real time operations. Paul & Aslan (2021) was able to successfully apply contrast limiting histogram equalisation in a real-time face detection program but with low resolution images. This may indicate that while effective, contrast limiting histogram equalisation may only be used in applications which utilise low resolution imagery or poorer hardware.

4.2 Artificial intelligence

Artificial intelligence (AI) is simply the ability for a machine to perform an action intelligently, where intelligence is defined by a human standard of intelligence and a machine can therefore be seen to perform an action to the standard surpassing or equaling the ability of a human. AI's use in zero sum games is a popular area of study among academics and countless publications have focused on the role of AI in chess. A zero sum game can be defined as a competitive game played by two or more players where the sum from the reward for winning is proportionate to the sum of the loss from losing. Translating this into a draughts game played by two players, there is a 1:1 payout ratio where if a player takes an opposition piece they increase their total pieces while reducing the opposition's total pieces. Implementations of AI in zero sum games utilise tree search algorithms to find an optimal move that maximises the player's reward while minimizing the opposition's reward, the most common technique is Alpha-Beta pruning.

5 Design

This chapter documents the design of the computer vision project to be undertaken. Specifically some key concepts relating to the design of the systems, project requirements and methodologies employed. This section also provide an overview of quality controls adopted to ensure the project output is aligned to the requirements of this project.

5.1 The rules of English Draughts

Internationally English Draughts can be seen as a variation of the classical game Chequers, which is also known in the Americas as checkers. Like chess, the game is played on a 8x8 gameboard, with two players and sixty-four evenly sized game squares separated into two distinct colours, with two sets of counters of different colours. Unlike chess, each set of counters are homogeneous with each other, where there exist only one type of counter, a draught, that is coloured classically either brown or white, red or black, black or white but there are also many alternative variations available.

Draughts, as previously mentioned, is a zero sum game where the objective of any player is to take as many opposition pieces as possible while minimizing the number of owned pieces that are taken by the opposition player. Each player in the game is bound by the sames rules that govern all aspects of the game, meaning each player can move and take opposition pieces in exactly the same manner with no deviation in rules based on colour of a draught.

In a game a draught can represent either a man or a king. In classical draughts a man is a single draught while a king is represented by two draughts stacked on top of each other, though within this project an alternative approach is taken which is detailed in the implementation section of this project.

Each player starts the game with 12 draughts, columns in the gameboard are positioned from one to eight and from left to right. The first draught of each player is positioned in column one in the first row that faces the player, the next draughts are then placed by alternating between skipping a square and then placing a draught in a square until twelve draughts are placed between three rows for each player as can be seen in the figure 11 below.

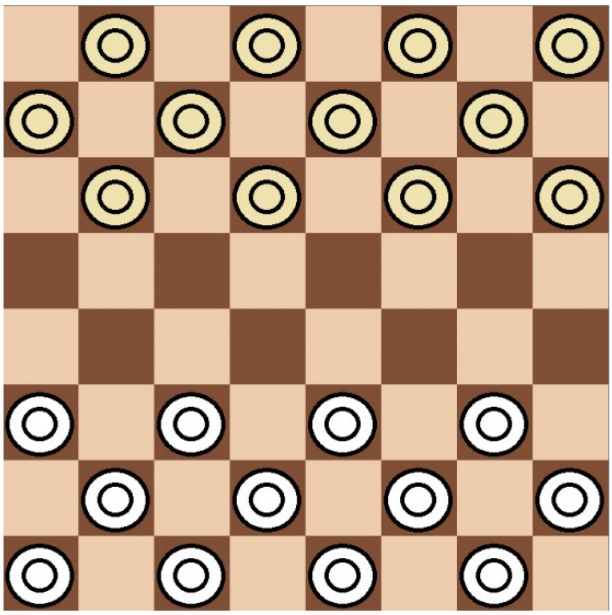


Figure 11: English Draughts game

A draught dedicated as a ”man”, may move one space diagonally in the direction of their opponent’s starting line, the first row facing the opponent. A man cannot move backwards towards the player’s starting line or in any other direction. The single objective of the game is to ”take” an opponent’s draught, this involves the player’s draught jumping over the opponent’s draught and removing it from the gameboard. Once all of the opponent’s draughts are removed from the board or the opponent is unable to make a move the player wins, conversely, if all of the player’s draughts are removed from the board or the player is unable to make a move the opponent wins. In cases where there are no further moves available to either player, a draw is decided. A player can take an opponent’s draught, both man or king if, an opponent’s draught is positioned diagonally adjacent to the player’s draught, the opponent’s draught is between the opponent’s starting line and the player’s draught and finally, if there is an unoccupied adjacent square that is diagonally behind the opponent’s draught that lies in the direction of a diagonal move by the player. If either an opposition draught or the player’s draught blocks a diagonal move behind the opposition’s draught, then the move is not possible. In cases where a player takes an opponent’s draught and from this new position is then able to take another opponent’s draught they must do so, even if it puts the player into a position of disadvantage, the player must continue to take the opponent’s draughts until there exists no further available moves.

When a player’s draught reaches the starting line of the opposition it becomes a king. The king is crowned by placing a draught on top of the original draught. The draught must wait a turn before moving before it is able to move. A king differs to a man as a king may

move in any diagonal direction either forward or backwards by one place. The king may also take an opponent's man or king in any diagonal direction. Apart from the direction of movement, the king then functions like a man and is bound by the same rules.

Game alterations

While this project follows the same game rules as English Draughts in the game-logic, 4 separate coloured counters are used represent player man, opposition man, player king and opposition king. This is due to two reasons. Firstly, the supplied counters with the board have similar colour patterns as squares on the board and when subjected to different light levels, may affect the accuracy of proposed computer vision process. Finally, as this project identifies counters by colour it will be impossible to segregate counters representing men and counter representing kings. It is therefore mandatory to replace the original counters for this game.

5.2 Agile methodology

This project documented in this work takes an agile approach to project implementation. An agile approach takes a large project and breaks it down into sub-components, for each component a sprint is completed. A sprint can usually be deconstructed into a planning stage, design stage, implementation stage and a testing stage with a final review. A team applying agile in their project are able to provide a value added benefit to their end customer a lot quicker than if applying other methodologies such as waterfall project management. After each sprint a working element of a project is designed, providing value for the customer. With each consecutive sprint, additional functionality is provided to the current project, continually improving a project by delivering early and often, unlike a waterfall model. The design of the end system produced in this project is built by combining a series of smaller processes and functions, it is then sensible to adopt an agile framework as the project can be divided into many smaller tasks, making the benefits of working with an agile framework extremely obtainable.

The choice of an agile project framework is in line with reference to other works. Wu et al. (2019) applied an agile project framework in the implementation of facial tracking computer vision project which was assisted by CNN deep learning. Wu et al. (2019) praised the flexibility an agile project frameworks offers to large computer vision projects with many components.

5.3 Project design

The key sections of this project are, computer vision, system logic and NAO hardware. This structure can be witnessed in Appendix A.4 which contains the file structure of this project. This project takes an object orientated approach to design, with the project design being flexible and agile, allowing easy scalability for future work and clear visibility in source code. Appendix A.3 provides an overview of the process stages in playing a draughts game which also includes details of all key actors, their responsibilities and functions within delivering the game with NAO.

Reviewing figure 12, this project will first acquire an image using the integrated camera on the NAO unit. Next the project will apply some image processing techniques including: imaging resizing, gamma correction and Gaussian blurring to prepare the image for object detection. Next the project will apply Canny edge detection and contour detection to identify the boundaries of the gameboard object and the four boarding corners of the gameboard. Next a projective transformation will be applied to provide an aerial view of the gameboard exclusively. The image will then be partitioned into sixty-four evenly spaced smaller images which will be contained in a list, each image representing a square in the gameboard. A loop will then run through the list objects and apply thresholding to detect the highest white pixel count based on iterations through known RGB ranges of pixel values for draughts. The output of this operation will be a NumPy 2-D array of classified counters and their location within the board, this array will then be used as input into the draught's game logic and AI processes.

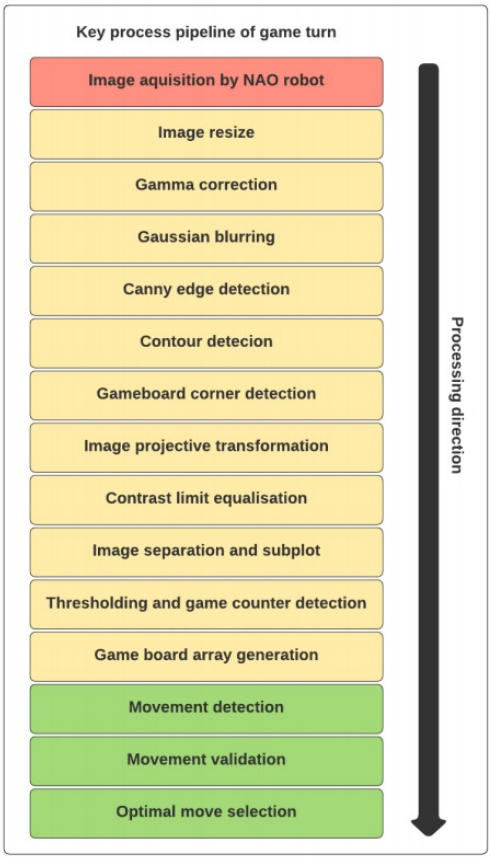


Figure 12: Project pipeline

NAO Specification

The NAO robot used in this project is the fourth version produced by Soft Bank Robotics. The utilised features of the NAO include its cameras and head joint functionality. This designed version of the robot included two cameras, one positioned on the forehead of the robot and the second is positioned directly on the chin of the robot, the exact locations can be seen figure 13. One considered weakness of the robot is the resolution of each of the cameras, the maximum resolution of each camera is 1280x960 which is extremely underpowered compared to classic computer vision projects that apply cameras with a much higher output resolution. Images with a greater count of pixels contain more details and make object detection clearer when applying operations such as corner detection or Canny edge detection. However, the clearer benefit of a lower resolution camera is quicker image processing due to a smaller image size, however due to a limitation in pixels, this may present more opportunity for noise disturbance. In addition to the cameras, the NAO robot makes use of joints and actuators that provide pitch and yaw functionality to the degrees of freedom specified in figure 13. The robot will apply head re-positioning in the image acquisition phase to ensure the entirety of the gameboard is captured in the processing image.

Minimising image computation time is an essential step when using the NAO hardware as the NAO only benefits from a 1.6GHz Intel Atom z530. The only computation expenses of the NAO robot will be however, the initial image capture, head re-positioning and providing speech commands. The additionally computational burden will fall upon the additional system used in this project which houses a 2.70GHz 4 core Intel(R) Core(TM) i7-6820HK CPU and 16GB of on-board memory.

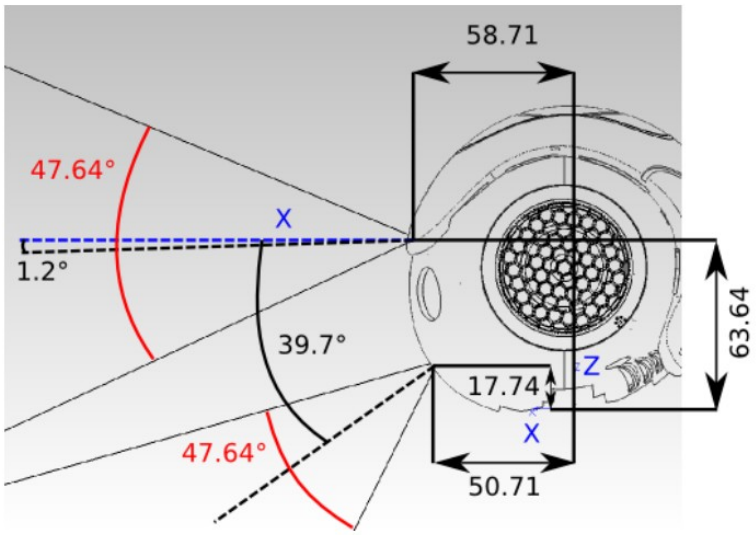


Figure 13: NAO head and camera positioning

Evaluation

Considering the hypothesis statement of this work, ”It is a viable application to utilise NAO robotics and computer vision, unsupported by machine learning, in detection and placement of the counters in the game draughts”, a viable solution for object detection in computer vision can arguably be a solution that is efficient in terms of speed of processing and accurate in object classification from detection. Additionally, this project also includes a third element of consideration which is the accuracy and validity of move allocation in the game of draughts. As this project is primarily focused on delivering a ”viable” computer vision solution for detection of game pieces and the game board, it is not mandatory to consider the correctness of game logic as this is a secondary consideration and if the NAO robot computer vision approach is assumed satisfactory, developing a better or more complex AI system for move decision and validation is trivial for those wishing to replicate the work. However for completeness, a secondary evaluation criterion is assigned for the critiquing of optimal move logic.

Primary targets for this project have been considered with regards to academic publications to ensure competitiveness and robustness from the proposed computer vision solution.

Time taken for the processes of image acquisition, image processing and gameboard array generation will be measured against a key performance indicator (KPI) of 10 seconds. This is inline with Li et al. (2019) that implements a body position identification system with a CNN deep learning position identification system, that achieved a three second processing time. Unlike a CNN model, the image processing in this project utilises no training models and therefore, benefits from a time saving in project setup and initiation, justifying the slightly more lenient time KPI. In a subjective view, it is also acceptable to wait 10 seconds to detect a gameboard and play a move, in a game that at late stages is played exponentially slower.

Another primary KPI target for this project is accuracy in detection of game board counters. The KPI target for this will be to maintain an average counter classification accuracy of 85%. This target is constructed in accordance with the work published by Wölflein & Arandjelović (2021), who were able to achieve a classification accuracy of 90% of chess pieces in a game utilising a CNN deep learning model, making 85% within an acceptable accuracy for classification. Finally this project will establish a secondary objective of maintaining an average optimal move decision accuracy of 90%, an optimal move being defined as a move that maximises the amount of player draughts taken in any one move by the AI. As the decision of optimal moves is not a reflection of computer vision, the arbitrary target of 90% will reflect a strong prototype system that, if assuming computer vision KPIs are met, would provide a clear foundation for further interesting work in optimising AI logic.

Testing

As this project has followed an agile delivery methodology testing has be at the heart of all produced work. In each sprint cycle testing has been employed to ensure work to quality standard. The main methods of testing employed consisted of individual unit testing when deploying new software and black box testing in the test phase after sprint implementation. These testing solutions proved critical in delivering the final end solution.

6 Implementation

6.1 Nao image acquisition

To enable the NAO robot to acquire an image of the gameboard and game counters a connection from the host machine must be established to the robot by a WiFi connection. Next the robot must be positioned in close proximity to the board. Additionally, as the robot is

elevated over the board, the actuators in the unit head must be adjusted on the pitch and yaw axis to allow the robot to focus onto the board. This can be seen in figure 14 below.

Figure 14: NAO robot game positioning

In applying the specified functionality to the NAO unit a developer has the option of using the Choregraphe suite, which is software designed by SBR to allow non technical users to build a timeline process of actions to be preformed by NAO through clicking and dragging premade objects and functions into a timeline class that is activated at the click of a button. Choregraphe also allows more advanced functionality by allowing custom source code to be added to the timeline and activated. The alternative to Choregraphe, and the solution adopted for this project, is to download the Python2 SDK that provides access to the NAO modules and functionality through requesting access to a proxy server that returns the specific modules and required module functions, as can be seen in figure 15 below. In this project there were four modules accessed through the proxy broker system: ”ALTracker”, ”ALMotion”, ”ALVideoDevice” and ”ALTextToSpeech”. After placing the NAO next to the board, the ”ALMotion” module is accessed through the proxy and all actuators in the NAO unit have torque limitation removed, ensuring maximum movement speed. Next the ”ALTracker” module is called and the, factory set, automatic facial tracking system is deactivated to stop the NAO unit re-positing the head actuators if a face is detected while focusing on the game board. Next, the head pitch is adjusted to 0° which directs the NAO unit to face forward and the yaw is adjusted to 28.5° which directs NAO robot to look down, the degrees of movement available to unit head can be seen in Appendix A.5. Finally, module ”ALVideoDevice” is accessed and an image is captured with the attached

camera modules which is then saved to the project directory.

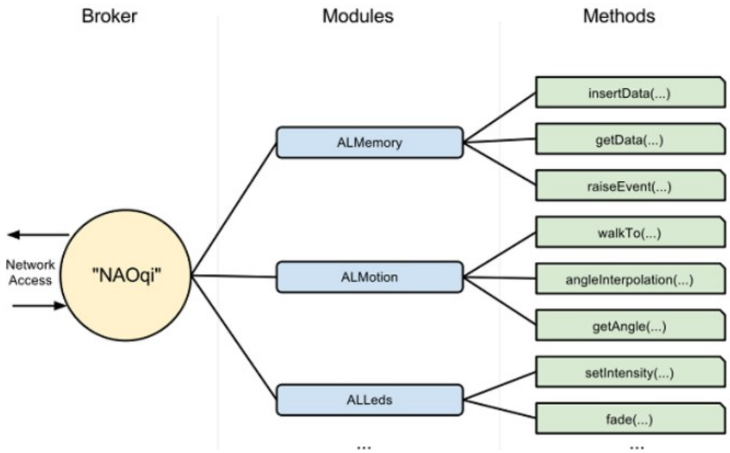


Figure 15: NAO API broker system

Soft Bank Robotics (2021*a*)

6.2 Image processing

The image acquired through the NAO unit is firstly resized using OpenCV resizing function from the native resolution of 480×640 to a 512×512 image and copy of the image is created, which is an essential step for evenly dividing the projective transformed image in later stages. Next the image is converted from BGR to greyscale using an OpenCV colour conversion function, which applies an greyscale conversion by averaging. The converted image then undergoes some gamma balancing, to help remove shadow areas and improve contrast. Gamma can simply be defined as the overall brightness within the image, where by an image with high gamma has many pixels with an average value close to 255 or conversely, has many pixels with values close to 1. Due to the positioning of the NAO robot, the gameboard edge the furthest distance away from the robot can suffer from increased shadow and additionally, so can game pieces positioned at distance to the robot. To adjust for low gamma pixels, pixels with a greyscale value of 0.65 or below were modified to 1. Image results can be seen in figure 16, it is clear that after gamma correction the shadow is removed from the board edge.

Figure 16: Left: image before gamma correction. Right: image after gamma correction

As the camera module attached to the NAO unit is only capable of low resolution images, noise is a particularly large problem for imagine processing. A 5×5 Gaussian blur kernel convolution operations was applied to the image. Looking at figure 17, the game board before the Gaussian blur was applied has excessive noise which can be seen in the empty white game squares. There also seems to be some salt and pepper noise across the entire image, which is likely to have been caused by the poor quality camera sensors. After Looking at the image after Gaussian blurring it is clear to see that the majority of noise has been removed from the white game squares , reducing the likelihood of incorrect identification of game counter and additionally, the presence of salt and pepper noise across the image is reduced.

Figure 17: Left: image before Gaussian blurring. Right: image after Gaussian blurring

Canny edge detection was then applied to threshold the greyscale image and outline edges. The optimal value for the maximum and minimum intensity gradients used within

the algorithm where trialled with many variations, which can be witnessed in Figure 18. Ultimately, it was a minimum intensity gradient of 95 and a maximum intensity gradient of 290 that was the optimal combination. It can be seen that the border of the gameboard is clearly defined and the edges that are contained within the board are ignored at this minimum and maximum value, which is required for further image processing steps. In Canny edge detection with lesser maximum values there is significant edge detection inside the gameboard that looks to be caused by noise. At higher maximum values, the definition of the gameboard borders are reduced.

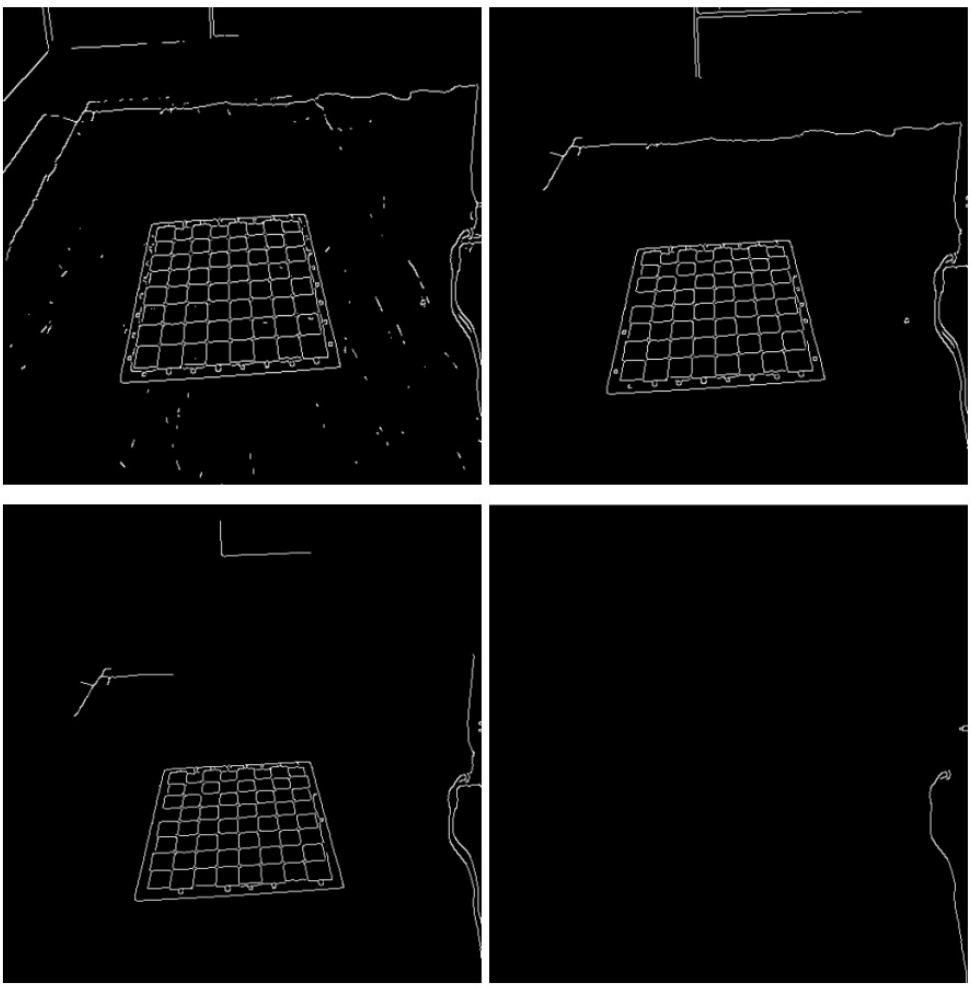


Figure 18: From left: gradient(Min:95 Max:100), gradient(Min:95 Max:190), gradient(Min:95 Max:290), gradient(Min:95 Max:390)

Next contours are detected from the image adjusted by Canny edge detection and polygon approximation is applied. An array of all contour points is returned, looping through each list of points, the contour is polygon approximated by a factor of 1%, this produced the most accurate points for the outline of the gameboard. The approximated points have been drawn onto the image in Figure 19 below. As the object of game board detection is to find the corner points of the board, for each set of polygon points that contain four or

more points a gameboard top left point, top right point, bottom left and bottom point is established. This is done by first finding the bottom right corner and the top left corner, where the criteria for the bottom right corner is:

$$\max \sum (x + y)$$

And conversely, the location of the top left corner is given by the criteria:

$$\min \sum (x + y)$$

The remaining corner points are then calculated by finding the points furthest away from the other reference points within a acceptable deviation, which is given as 80 pixels. To determine the optimal corner points for the gameboard from the lists of new potential corners, the area between the suggested points is then calculated. If the area is between 25000 and 70000 pixels it is likely to be the corners of the game board, any small or large objects are then ignored. The optimal points are then returned as a list.

Figure 19: Poly approximated contours

Next a transformation matrix is calculated using OpenCV getPerspectiveTransform() which requires the dimensions of the original image and the new dimension which are the corner locations of the gameboard. The matrix is then used to calculate the image warp, with the final result providing and aerial view of the gameboard. Next the image is equalised using local contrast limiting equalisation and the image is also cropped to remove the gameboard borders. This leaves only an image of all the game squares. The products of these processes can be seen in figure 20.

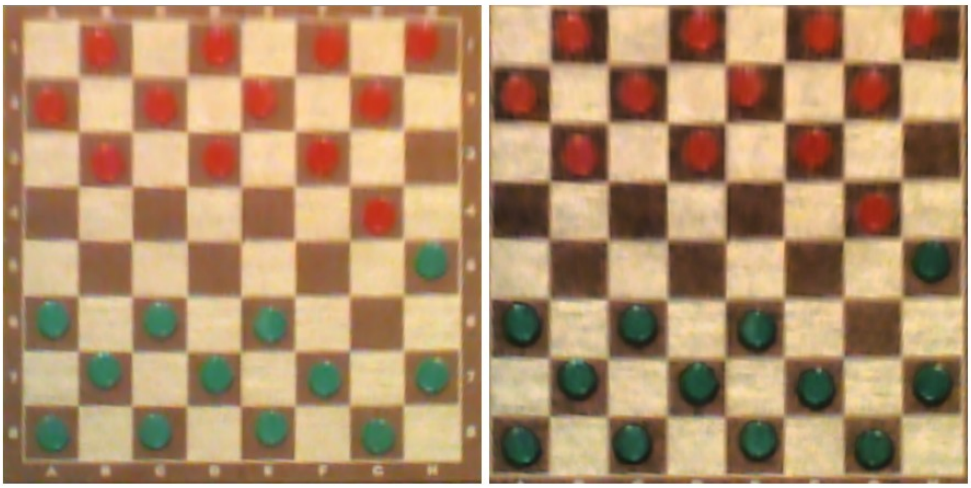


Figure 20: From Left: No image equalisation, image equalised and board bezel removed

Finally the transformed gameboard image is divided into 64 sub images and collected in a list. The list is then looped through, with each iteration, an additional sub-loop loops through know HSV ranges for game counter colours and applies a threshold operation to the image. To filter out noise, if an image contains 143 or more white pixels a counter should be located in the image. Additionally, if a image contains 3,481 or more pixels, the game piece is considered undefined as this is likely to be caused by a luminosity variance. Additionally, an erosion operation followed by a dilation operation is applied with a 5x5 kernel, this is to remove small amounts of isolated noise. When changing the HSV ranges if the count of white pixels in the image is the greatest, the image and the game square is considered occupied by a counter. A NumPy arrays is created to store the location of the counters and the counter type. An image of a virtual gameboard is then created based on the NumPy array and the arrays becomes the input into draughts game logic. The final output of the computer vision process can be seen in Figure 21.

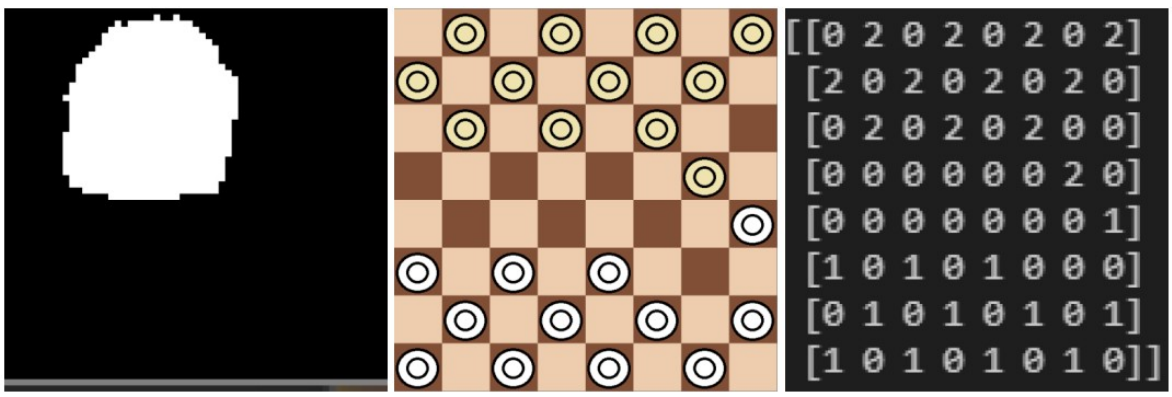


Figure 21: From Left: Counter detection, rendered virtual gameboard, created NumPy array

6.3 Draughts game logic

The game script is run from a main game loop. A game begins with the NAO robot capturing an image. Image processing steps are then applied and the NumPy array and virtual gameboard are generated that are the input into the game logic. The user then must validate if the array generated is accurate, otherwise the process step is repeated. Next another image of the gameboard and a new array, is generated with the NAO unit and compared against the original NumPy array. To understand the available moves in the game, the function ”get valid moves”, calls the functions ”traverse left” and ”traverse right”, these two functions use recursion and backtracking to find all available moves from a selected draught piece. A visual of this code can be seen in Appendix A.6. Considering ”traverse left” with a player counter, the function begins by considering if a move is immediately available above and diagonally left, where there is space or there exists an opponent counter. If there is space, the function adds the position to a list of available moves, returns the list and backtracks, next calling the ”traverse right” function. If an opponent counter is located the function is then called recursively to check if the next immediate space behind the opponent counter is occupied or available and this process continues until all the available moves are found. The list of available moves is then looped through, if the new move location of the player's counter is found in the list of available moves the NAO robot verbally confirms this to the player, if the move is invalid, the NAO robot also verbally confirms this to the player and the cycle restarts until a valid move is played by the player.

When it is the computers turn to decide a move, the function ”get opp move” is called to determine the best AI move. A list of all opponent counter locations is looped through, each location is input into the ”get valid moves” functions and list of available moves for the corresponded piece is returned. Then an optimal move for the AI is decided based on the maximum count of moves taking in one turn. For example, given the choice of moving down to the next available space which is diagonally left and immediately below the opposition counter, or making two moves by moving diagonally right and moving the counter to two places by taking 2 player counters, the latter options will be chosen. In the case where the AI has the choice of moving only one space, left or right diagonally, the optimal move will be decided based on the number of rows moved, as more rows moved signifies the AI has taken a player counter. This code can be seen in Appendix A.7. The NAO robot then verbally instructs the player to move the opponent counter, once move the player confirms the move is complete and a new image of the board is taken to ensure the correct move-

ment. If the move is performed incorrectly the NAO robot re-instructs the player and cycle repeats, if the move is correct NAO robot verbally confirms this with the player and the player's turn process begins. When there are no available moves by either side or the count of counters is 0, the game ends.

7 Results

This section of work provides overview of the results obtained by the composed computer vision system and draughts game logic. The results are also measured against the KPI targets established in the project design.

7.1 Processing times

The key measure of success in this project lies balanced between the image process times and image classification accuracy. A sample of observed processing times were recorded from two different start points in the computer vision process. Firstly, the time taken to connect to the NAO robot by wifi connection, generate a new image of the gameboard, process the image and detect the board and the location of game counters and finally return the computer NumPy array. In addition to this, the time taken from processing the image and generating the NumPy array exclusively was also recorded. The results of these observation can be seen in figure 22 below.

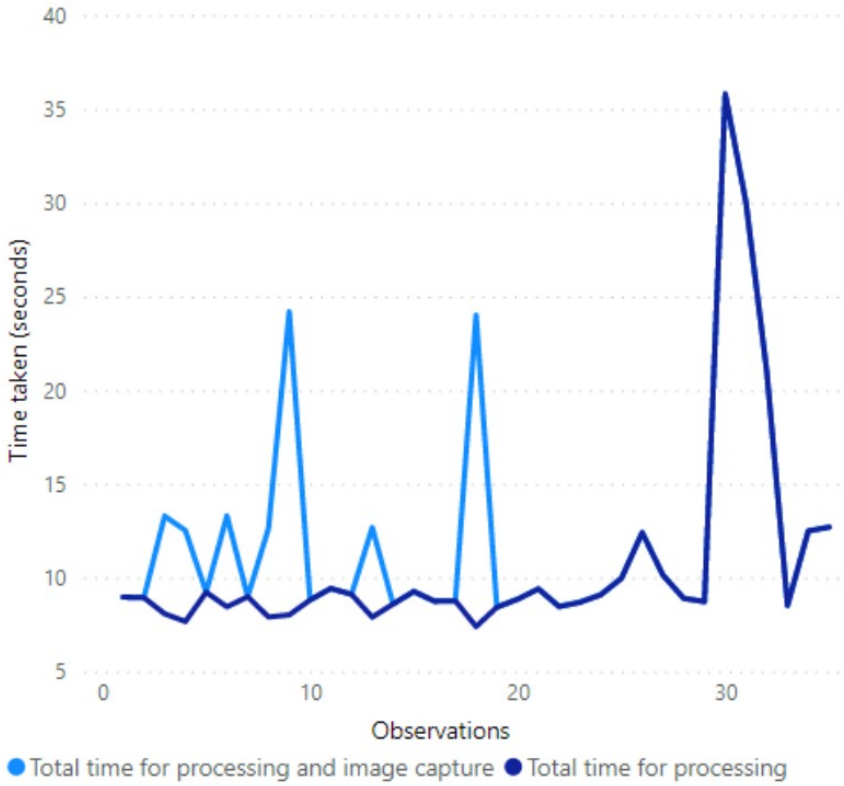


Figure 22: Time taken to complete computer vision processes

Looking at the results in figure 22 to it possible to see a series of interesting results. Firstly, observing the time taken exclusively for processing it can be seen that process times are consistent for the majority of observations, in most cases being close 8 seconds or less to load the image object, process it, detect board corner, identify the counter types and return the created NumPy array. However, closer to the end of the observations a sharp spike in increased processing times are noticed, with times taking in excess of 30 seconds and in one observation over 35 seconds. When measuring these processes, natural light in the room was suddenly reduced due to a change in the time of day, this changed the contrast and luminosity levels within the room, the position of the shadow cast be the NAO robot also changed compared to the previous observations. This affected the ability of the computer vision to find the board game corners. In the final observation, light levels changed to match the original conditions in the room and observed results were closed to early observations. The total time from connection to the NAO unit, image acquisition, image system upload, computer vision processing and NumPy array generation was also measured, which can be seen as the full computer vision process from input to output. The documented results are slightly sporadic, where in some observation there is little to no deviation from the total processing times, in other cases there exists much larger deviations in time taken to

complete the full process from image collection to array output. A majority of these time deviation were caused by connectivity issues with the NAOqi API system, making the camera functionality inaccessible on the NAO unit. Additionally, using the python SDK it was noticed that some module functionality can be inconsistent, for example, the NAO robot may adjust the actuators in the head outside the angles specified by the written python script due connectivity issues. This in turn, caused images to be taken by the robot when the camera was not positioned correctly and the board could not be observed. Overall, these occurrences has a significant affect on the average time taken for the majority of observations. The average time taking in the process operations exclusively was 10.79 seconds and if the outliers, caused by the sudden natural light level changed, are ignored an average time of 9.1 seconds is achieved in processing. This time is below the KPI target set of 10 seconds or above the KPI target of 10 seconds if outliers are included. The average time taken for the complete process from image acquisition to array generation was 12.43 seconds, it can be seen that the total time take for the aggregate process was on average, 1.63 seconds slower than just the image processing stage alone. Based on the sample data, there exists a 20% probability that a time of processing exclusively is observed of over 10 seconds. There is also a probability of 40% that the time taken to connect to the NAO, acquire the image, process the image and return the NumPy array is over 10 seconds.

7.2 Image classification

In measuring the accuracy of classification of draughts through computer vision, the counter classification algorithm was run 60 times with the accuracy measured in percentage of game squares identified correctly, meaning the generated NumPy array correctly maps the gameboard envisioned with the location of every counter, counter type and empty space correctly. The first 30 observations in this work use only the player and opposition draught and exclude the king counters. Additionally, to test the robustness of this process, the gameboard was moved by small increments and positions of draughts on the board were moved.

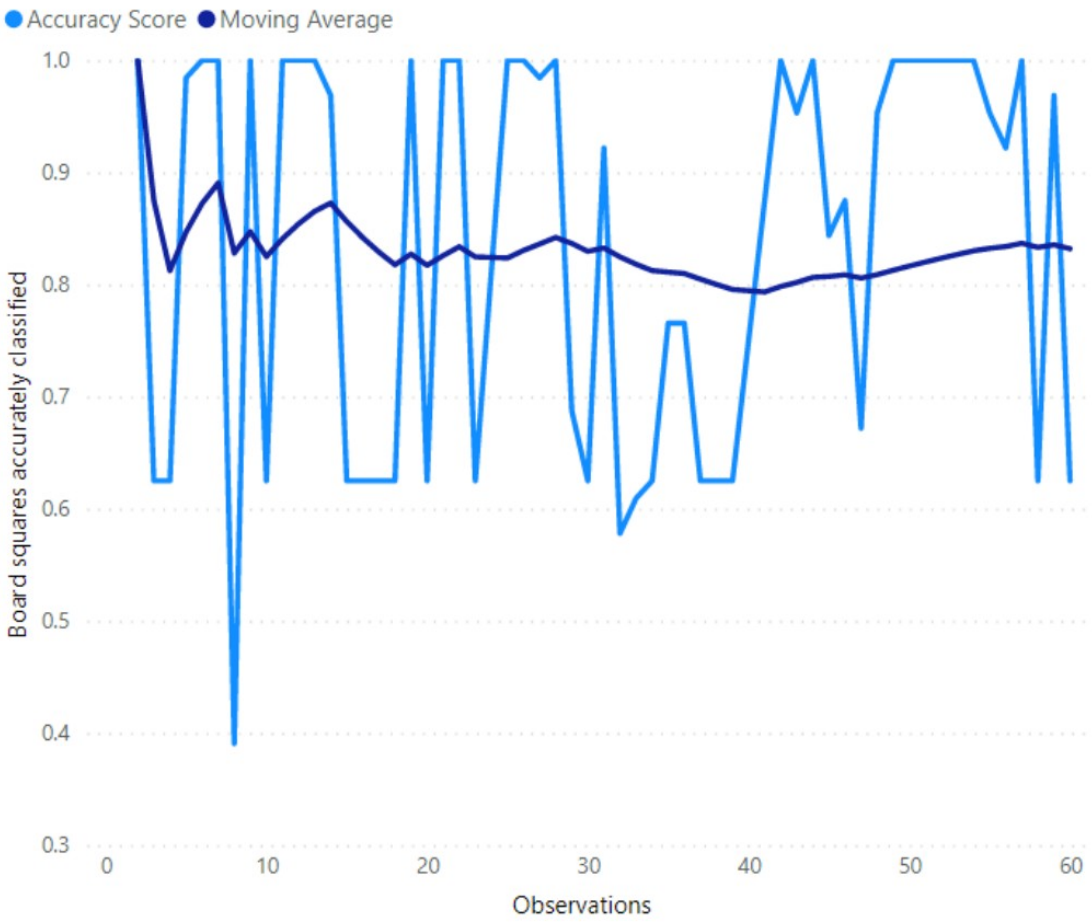


Figure 23: Measure of image classification accuracy

Looking at figure 23 it is possible to see there is extreme volatility in the accuracy measures collected. There are large peaks in accuracy followed by troughs. Interestingly in most cases, the results are either completely accurate or close to an accuracy of 60%. In the measurement process only 24 counters were ever on the gameboard at anyone time, the 60% scores can be attributed to this, no counters were identified and the algorithm assumed all squares were vacant. Looking at the count of observations with an accuracy close to 60% is evident that an empty gameboard was visualised many times by the computer vision algorithm. However, there is a single score below 40% which indicates cases where empty squares and counters are identified incorrectly where nearly non-existence. In processing each image, there were a significant amount of times when the NAO robot would position the camera module at an angle unspecified by the script and the position of the gameboard as well as surface reflections on the game counters would change in the captured image. Additionally, there were changes in light levels throughout the measuring process that may have contributed to significantly to accuracy scores. The majority of observations did however accurately identify all game counter types and locations as well as the locations of empty squares. Overall an average accuracy of 83% was obtained in gen-

erating the NumPy array from the gameboard image. This result fails to exceed or equal the target KPI of 85% in image classification.

7.3 Game logic

The game logic was measured by defining the best move, which is the move that maximises the number of pieces than can be taken in one turn, measured over 5 separate games. Noticeably, the AI chooses a move that does maximise the number of player counters obtained in a move. When there is the option to take a player counter or two player counters, the AI will always chooses the latter move that maximises the number of taken counters. The AI is not intelligent however and will often move an opposition counter to a space that makes it directly obtainable on the next turn by the player. Additionally, the AI moves become easily predictable as when all available moves are evenly scored, the AI will always move to the next available draught that is closest to the maximum boundary of the gameboard. However the rules of English Draughts is still maintained, as the AI confided to the rules with 100% accuracy and will take all player counters immediately available to them. A optimal move obtainment rate as defined by criteria established above is 100%. This is 10% better than the KPI target of 90% accuracy.

8 Evaluation

In this section the results from the implementation of the proposed system are considered with a key focus on computation cost, efficiency and feasibly. With the support of academics references, it possible to test these key considerations to enable confirmation or reject of the hypothesis statement, given as, ”it is a viable application to utilise NAO robotics and computer vision, unsupported by machine learning, in detection and placement of the counters in the game draughts”.

8.1 Is the proposed system viable?

Firstly considering the main objectives of this work, which was to obtain a computer vision system that is ”viable”, not optimal, to play the game of draughts using the camera module equipped to the NAO unit. The obtained accuracy in determining the correct type of counter in a game square and the location of unoccupied squares was close to the desired KPI accuracy, competitively set according to similar academic work, of 85%, falling short

with an obtained accuracy of 83% due mainly to inconsistency with the NAO unit and the positioned angle of the head actuators and the additional problems due to light variance. The accuracy of classification could likely be improved by further optimisation of the computer vision algorithms, which would could be tailored to a specific scenario where there are fixed conditions in terms of light, NAO location to the board or a stationary camera. Such scenarios are likely to be rare to find and in most cases it is both more flexible and convenient to apply a computer vision solutions that has a greater level of versatility, such as a system supported by machine learning. Considering that NAO benefits from having a large range of movement which makes it extremely desirable for projects involving robots compared to other alternative robots that are more specialised and limited, applications involving NAO need to be versatile and adaptable. However, it is not prudent to consider the proposed system in this work irrelevant for classification and object detection operations. The OpenCV framework and concepts applied in this computer vision solution are very low level and accessible to many new and inspiring programmers. Making improvements to the proposed system should be easy given time, which is not a benefit of alternative computer vision systems that use complicated CNN solutions.

This approach is extremely viable with consideration to time constraints. A CNN solution for example would require a significant proportion of time spent on training the system in various light conditions, especially when we consider a board could be rotated, there may be many shadows, sensor interference or surface reflection which required further training time to account for. In the solution proposed it takes a matter of moments to load a live camera feed with a slider tool to adjust attributes such as threshold levels, gamma correction, canny edge detection boundaries and many more features to enable improved image detection and classification. There is however a noticeable weakness with consideration for time. The system proposed had an average image processing time of 10.79 seconds which fell just outside of the KPI target of 10 seconds, which is again established based on literature. When also factoring in problems with image acquisition through NAO, the total time taken between image acquisition and generation of the NumPy array, which contains the gameboard positions, is on average 12.43 seconds. This is worryingly long when we consider Li et al. (2019) was able to process an image in 3 seconds with a CNN.

When the proposed computer vision solution is combined with the proposed game logic there exists a game of draughts which provides an available opponent, that follows the rules and will take the player's counters whenever possible, 100% of the time. The AI is not however, an intelligent or intimidating opposition and would not prove to be a formidable

opponent. This however could be easily changed with work to integrate Alpha-Beta pruning for a more intelligent and engaging opponent. This is definitely an afterthought however, as it is a more exciting prospect to understand the capabilities of NAO and the easily accessible computer vision algorithms and libraries that exist, which can be used to create interesting and diverse work that is centred on NAO, an exciting form of technology. It is worth noting that using NAO has induced much challenge. The API proxy system is not easy to understand or use with Python. There is little to no documentation available online that demonstrates the basic functionality of NAO programmed with Python, but for programming with C++ there is a reasonable amount of documentation to follow, potentially outlining why full documented processes of NAO programming implemented with Python are rare to find. Additionally, the fact that the NAO Python SDK can only be used with Python2 does also reduce the amount of opportunities to integrate new technology from Python3 modules or future modules. Issues with consistency may be the key consideration with using NAO, the API does occasionally fail to call correct modules and connectivity is an issue when using the API, compared to an offline solution. These potential issues could cause programs to crash, making NAO not usable in any sensitive or essential processes, which realistically is an unlikely application of NAO technology anyway.

8.2 System comparison to literature

The proposed system of this work does have several other benefits over similar work produced by academics. The choice to use computer vision to detect counters in draughts is significant when compared to work such as that published by Kopets et al. (2020), who used magnetised counters in a game of checkers. This technique is very restrictive if you were to play with multiple opponents using different gameboard in one sitting or wish to take the foundations of your work and apply it to other games. The proposed solution in this document has a greater level of adaptability and the foundation available to a programmer that can be modified and applied in other games such as monopoly, tic tac toe, battleships and many more popular games. Though there are certain limitations to this solution. Wölflein & Arandjelović (2021) was able to provide real-time image processing in determining the location and type of chess piece in a game of chess. This obtainment even after much process optimisation would just not be possible for the solution proposed in this work, the system foundation would certainly not be able to extend to games such as chess, but is that really necessary. In addition Wölflein & Arandjelović (2021), many

academics such as Szemenyei & Estivill-Castro (2018) have explored complex CNN solutions for computer vision which excluded the novice from replicating, when a key benefit of NAO is accommodating easy and accessible code to introduce beginners to robotics, this is why Choregraphe software, produce by SBR, was released. With the solution identified in this work there are clear benefits in building quick and effective systems for small tasks, many other academics appreciate this benefit too, such as Magallán-Ramirez et al. (2021) who took a similar approach with the use of Canny edge detection in identifying a correct path in a path-finding algorithm.

9 Conclusion and Further work

After reviewing the achieved results from the implementation and testing of the computer vision solution suggested in this work the hypothesis statement can be considered proved. This is because with a slight amount of optimisation, all KPI targets could be achieved, that is an accurate computer vision system that processes in a quick enough capacity to calculate and instruct moves in the game of English Draughts with the potential to plan moves intelligently all while using the NAO robot. However, the benefits of a NAO robot are not in the ability to specialise in a single task but rather in the ability to adapt to be used in many less complex tasks. Further work should be focused on reviewing the approach taken in this document and optimising or altering the various process steps. It would then be extremely interesting to see the new adjusted work compared directly to the results of a similar implementation using CNN and the NAO robot. Why this may not be groundbreaking, the NAO robot has demonstrated its capabilities as a useful and adaptable tool when applied with computer vision. It is likely that through documenting more processes and creative applications using NAO, an omission of usable guides to implement projects with NAO will be quashed and a new generation of young programmers will provide growth to the field of computer vision with the creation of a plethora of innovative solutions developed with NAO.

References

Aarthy, M. & Sumathy, P. (2014), 'A comparison of histogram equalization method and histogram expansion', *Int. J. Comput. Sci. Mob. Appl* **2**(3), 25–34.

Ahmed, A. S. (2018), 'Comparative study among sobel, prewitt and canny edge detection operators used in image processing', *J. Theor. Appl. Inf. Technol* **96**(19), 6517–6525.

Akarsu, B., Karaköse, M., Parlak, K., Erhan, A. & Sarimaden, A. (2016), 'A fast and adaptive road defect detection approach using computer vision with real time implementation', *International Journal of Applied Mathematics Electronics and Computers* (Special Issue-1), 290–295.

Canny, J. (1986), 'A computational approach to edge detection', *IEEE Transactions on pattern analysis and machine intelligence* (6), 679–698.

Douglas, D. H. & Peucker, T. K. (1973), 'Algorithms for the reduction of the number of points required to represent a digitized line or its caricature', *Cartographica: the international journal for geographic information and geovisualization* **10**(2), 112–122.

Grand View Research (2020), 'Computer vision market size share report, 2020-2027'.

Han, D. (2013), Comparison of commonly used image interpolation methods, *in* 'Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering (ICCSEE 2013)', Vol. 10.

IBM (2021), 'What is computer vision'.
URL: *https://www.ibm.com/topics/computer-vision*

Jayasuriya, S. J., Sampson, A. S. & Buckler, M. B. (2017), 'Reconfiguring the imaging pipeline for computer vision'.

Karuppiah, P., Metalia, H. & George, K. (2018), Automation of a wheelchair mounted robotic arm using computer vision interface, *in* '2018 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)', pp. 1–5.

Khemasuwan, D., Sorensen, J. S. & Colt, H. G. (2020), 'Artificial intelligence in pulmonary medicine: computer vision, predictive model and covid-19', *European Respiratory Review* **29**(157).

Kopets, E. E., Karimov, A. I., Kolev, G. Y., Scalera, L. & Butusov, D. N. (2020), 'Interactive robot for playing russian checkers', *Robotics* **9**(4), 107.

Lewis, D. & Bailey, D. (2004), 'A checkers playing robot', *Raport instytutowy, Institute of Technology and Engineering, Institute of Information Sciences and Technology Massey University* .

Li, C., Imeokparia, E., Ketzner, M. & Tsahai, T. (2019), Teaching the nao robot to play a human-robot interactive game, *in* '2019 International Conference on Computational Science and Computational Intelligence (CSCI)', IEEE, pp. 712–715.

Magallán-Ramirez, D., Rodriguez-Tirado, A., Martínez-Aguilar, J. D., Moreno-García, C. F., Balderas, D. & López-Caudana, E. O. (2021), 'Implementation of nao robot maze navigation based on computer vision and collaborative learning'.

Matuska, S., Hudec, R. & Benco, M. (2012), The comparison of cpu time consumption for image processing algorithm in matlab and opencv, *in* '2012 ELEKTRO', IEEE, pp. 75–78.

Meng, L. (2019), Applying Reinforcement Learning with Monte Carlo Tree Search to The Game of Draughts, PhD thesis.

Metz, A. (2021), 'Pour one out for pepper, the world's first humanoid robot, now 'discontinued".

URL: *https://www.techradar.com/uk/news/pour-one-out-for-pepper-the-worlds-first-humanoid-robot-now-discontinued*

OpenCV (2021), 'Histograms equalization'. [Online; accessed August 16, 2021].

URL: *https://docs.opencv.org/4.5.2/d5/daf/tutorial*$_p y_h istogram_e qualization.html$

Palekar, R. R., Parab, S. U., Parikh, D. P. & Kamble, V. N. (2017), Real time license plate detection using opencv and tesseract, *in* '2017 International Conference on Communication and Signal Processing (ICCSP)', pp. 2111–2115.

Paterson, J. & Aldabbagh, A. (2021), Gesture-controlled robotic arm utilizing opencv, *in* '2021 3rd International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)', pp. 1–6.

Paul, K. C. & Aslan, S. (2021), 'An improved real-time face recognition system at low resolution based on local binary pattern histogram algorithm and clahe', *arXiv preprint arXiv:2104.07234* .

Poda, X. & Qirici, O. (2018), 'Shape detection and classification using opencv and arduino uno.', *RTA-CSIT* **2280**, 128–36.

Prewitt, J. M. (1970), 'Object enhancement and extraction', *Picture processing and Psychopictorics* **10**(1), 15–19.

Shamsuddin, S., Ismail, L. I., Yussof, H., Zahari, N. I., Bahari, S., Hashim, H. & Jaffar, A. (2011), Humanoid robot nao: Review of control and motion exploration, *in* '2011 IEEE international conference on Control System, Computing and Engineering', IEEE, pp. 511–516.

Sobel, I. & Feldman, G. (1968), 'A 3x3 isotropic gradient operator for image processing', *a talk at the Stanford Artificial Project in* pp. 271–272.

Soft Bank Robotics (2021*a*), 'Nao api framework'. [Online; accessed August 21, 2021]. **URL:** *http://doc.aldebaran.com/1-14/dev/naoqi/index.htmlthe-naoqi-process*

Soft Bank Robotics (2021*b*), 'Nao head rotation range'. [Online; accessed August 21, 2021]. **URL:** *http://doc.aldebaran.com/2-1/family/robots/joints*$_r obot.html$

Statista (2019), 'Global robotics market revenue 2018-2025'.

Suzuki, S. et al. (1985), 'Topological structural analysis of digitized binary images by border following', *Computer vision, graphics, and image processing* **30**(1), 32–46.

Szemenyei, M. & Estivill-Castro, V. (2018), Real-time scene understanding using deep neural networks for robocup spl, *in* 'Robot World Cup', Springer, pp. 96–108.

Wölflein, G. & Arandjelović, O. (2021), 'Determining chess game state from an image', *Journal of Imaging* **7**(6), 94.

Wu, J., Wang, J. & Bai, Q. (2019), Design and research of intelligent idc computer room based on agile model, *in* '2019 International Conference on Robots Intelligent System (ICRIS)', pp. 349–351.

Xia, C., Fu, L., Liu, Z., Liu, H., Chen, L. & Liu, Y. (2018), 'Aquatic toxic analysis by monitoring fish behavior using computer vision: a recent progress', *Journal of toxicology* **2018**.

Xiaofeng, R. & Bo, L. (2012), 'Discriminatively trained sparse code gradients for contour detection', *Advances in neural information processing systems* **25**.

Yeotikar, S., Parimi, A. M. & Daseswar Rao, Y. V. (2016), Automation of end effector guidance of robotic arm for dental implantation using computer vision, *in* '2016 IEEE Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER)', pp. 84–89.

A Appendix

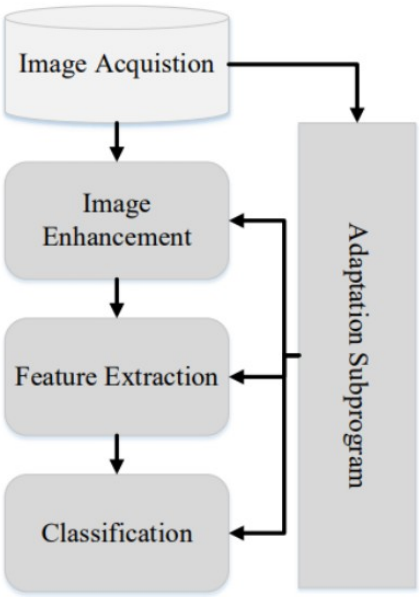


Figure A.1: Image processing pipeline
Akarsu et al. (2016)

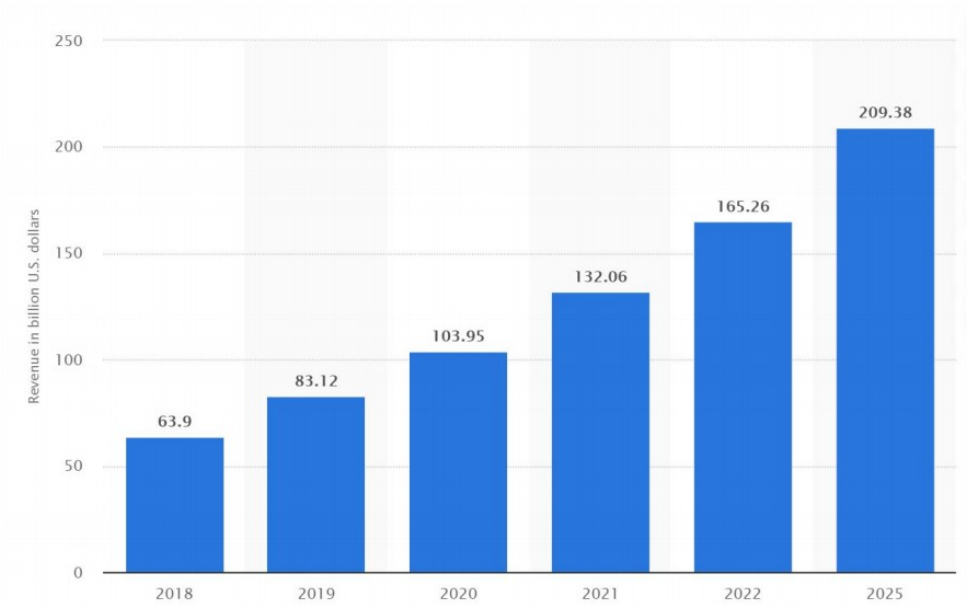


Figure A.2: Size of the global market for industrial and non-industrial robots between 2018 and 2025

Statista (2019)

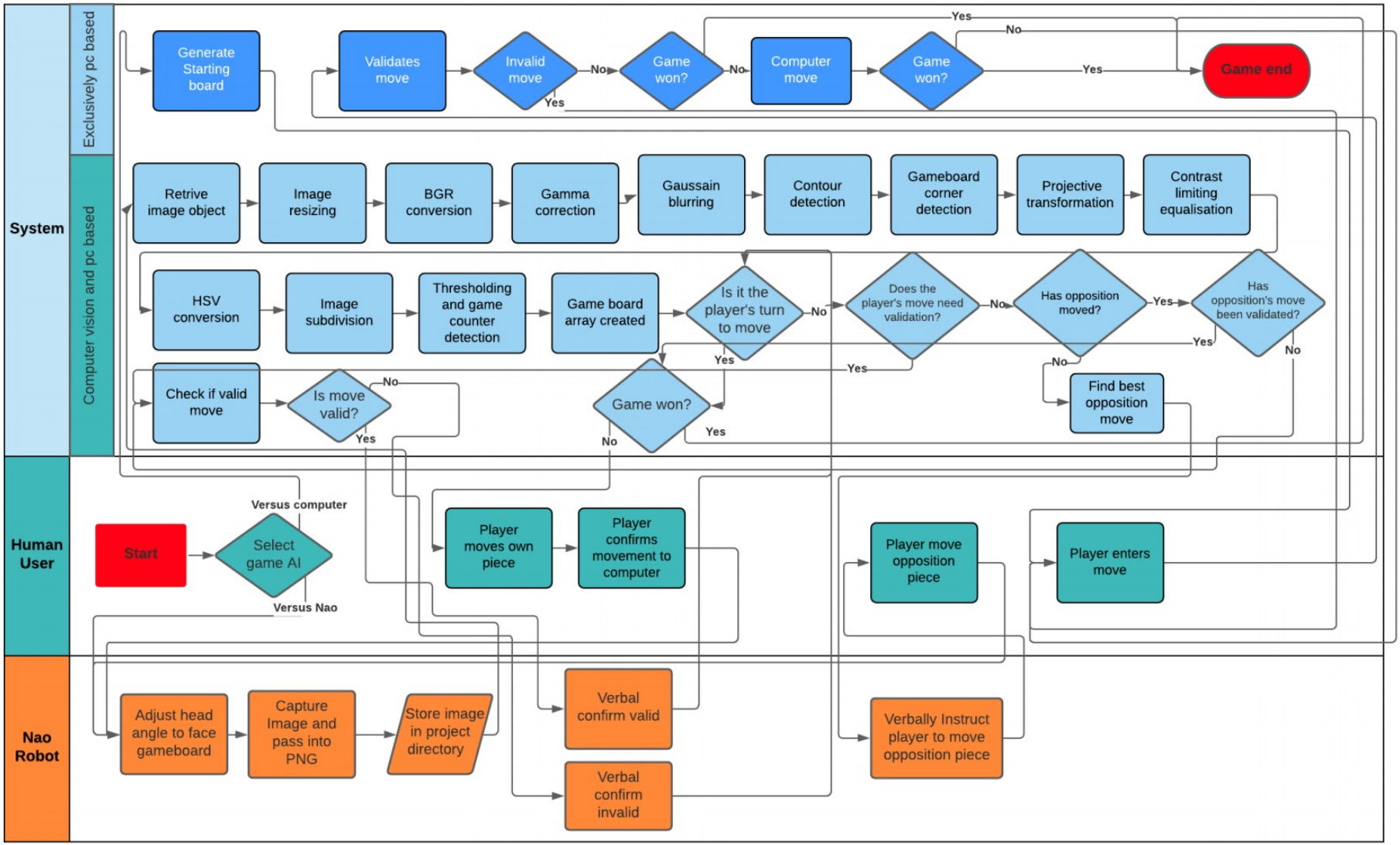


Figure A.3: Game process swimlane

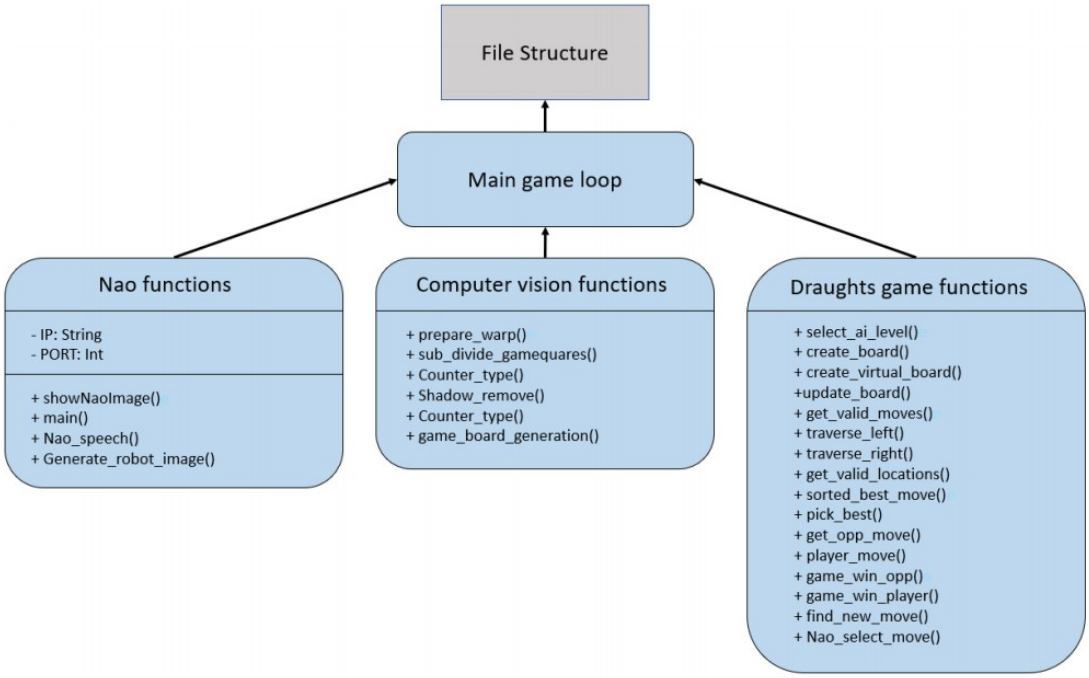


Figure A.4: Project file structure

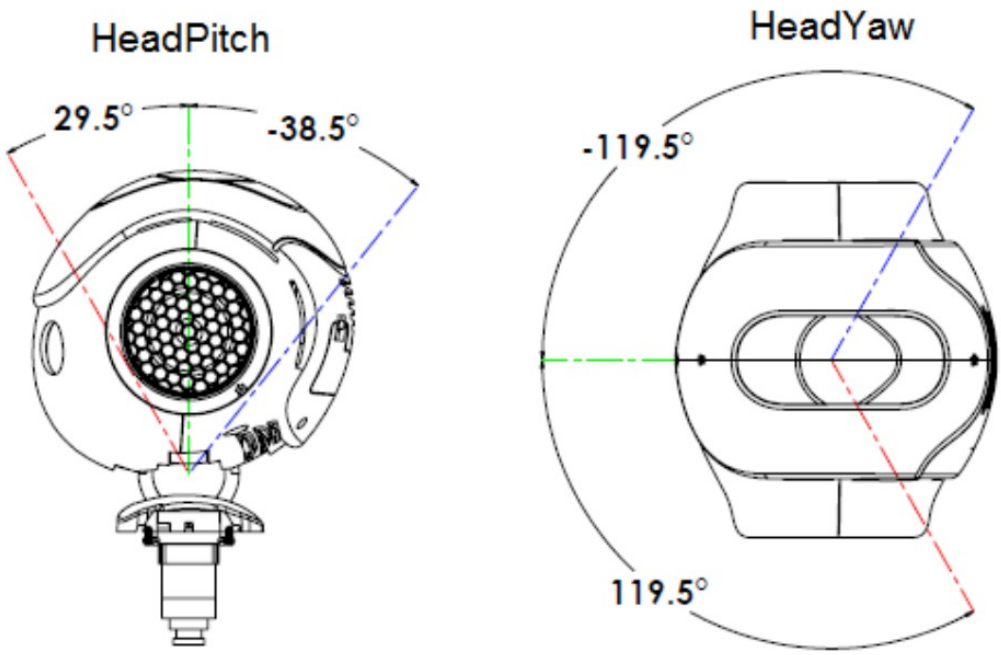


Figure A.5: NAO head rotation range
Soft Bank Robotics (2021*b*)

```python
#Gameplay mechanics
def get_valid_moves(piece, game_board, moves = []):
    #Store movement results
    moves = []
    board = np.copy(game_board)

    #Establish next column to move
    left = piece[2] - 1
    right = piece[2] + 1
    row = piece[1]

    #Recursively call movement
    if piece[0] == player_piece or piece[0] == opp_king or piece[0] == player_king:
        traverse_left(row-1, row-1, -1, piece, left, board, moves)
        traverse_right(row-1, row-1, -1, piece, right, board, moves)
    if piece[0] == opp_piece or piece[0] == opp_king or piece[0] == player_king:
        traverse_left(row +1, row+1, 1, piece, left, board, moves)
        traverse_right(row +1, row+1, 1, piece, right, board, moves)

    #Return the established list of available moves
    return moves

def traverse_left(start, stop, step, piece, left, board, moves, skipped=[], count=0):···

def traverse_right(start, stop, step, piece, right, board, moves, skipped=[], count=0):
```

Figure A.6: Code of available move logic

```python
def get_opp_move(board):

    #location of all available opponent counters
    locations = get_valid_locations(board)

    #Store the piece location and list of the best moves
    details_of_piece = []
    best_piece_to_move = []

    #loop through list of opponent counters, store move, find best counter to move
    for x in locations[0]:
        row = x[0]
        column = x[1]

        piece = int(board[row,column])

        piece_info = [piece, row, column]

        moves = get_valid_moves(piece_info, board)

        sorted_moves, highest_move = sorted_best_move(piece_info, moves)

        list_best_moves = pick_best(sorted_moves, highest_move, piece_info)

        print(list_best_moves)

        if len(list_best_moves) > len(best_piece_to_move):···

        elif len(list_best_moves) == len(best_piece_to_move) and len(best_piece_to_move) != 0: ···

    # remove player counter and change numpy array
    if len(details_of_piece) != 0:
        # print(best_piece_to_move)
        x, y = Remove_pieces(board, details_of_piece[0], opp_counters, player_counter, None, 0, best_piece_to_move)

    if len(best_piece_to_move) == 0:
        return [0],[0]

    return details_of_piece, best_piece_to_move
```

Figure A.7: Code of opponent move logic