Bacheloroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for IKT og realfag

Tommy Flåm, Jan Olav Skjong, Olav Andreas Grøtta

Digitizing a game of chess using image analysis

Bacheloroppgave i Dataingeniør
Veileder: Saleh Abdel-Afou Alaliyat
Mai 2021

NTNU
Kunnskap for en bedre verden

Tommy Flåm, Jan Olav Skjong, Olav Andreas Grøtta

Digitizing a game of chess using image analysis

Bacheloroppgave i Dataingeniør
Veileder: Saleh Abdel-Afou Alaliyat
Mai 2021

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for IKT og realfag

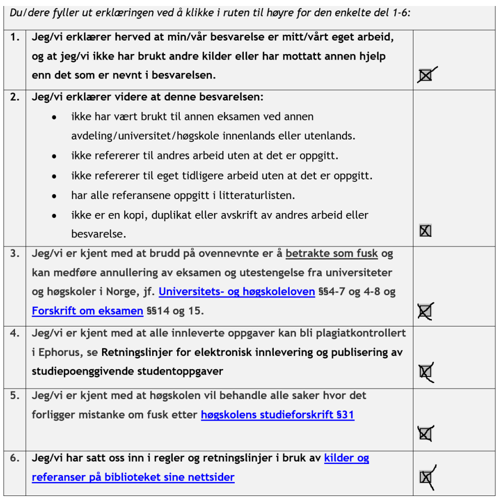
Digitizing a game of chess using image analysis

Tommy Flåm Jan Olav Skjong Olav Andreas Grøtta

May 21st 2021

Egenerklæring

Skjemaet vist under viser at alle tre stuentene i gruppen har satt seg inn i hva som er lovlige hjelpemidler, retningslinjene for bruk av disse og regler om kildebruk. Alle studentene er klare over hva som er deres ansvar i forhold til oppgaven og klar over hvilke konsekvenser som kan medføre av fusk.



Du/dere fyller ut erklæringen ved å klikke i ruten til høyre for den enkelte del 1-6:

1.	**Jeg/vi erklærer herved at min/vår besvarelse er mitt/vårt eget arbeid, og at jeg/vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.**	☒
2.	**Jeg/vi erklærer videre at denne besvarelsen:** • ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. • ikke refererer til andres arbeid uten at det er oppgitt. • ikke refererer til eget tidligere arbeid uten at det er oppgitt. • har alle referansene oppgitt i litteraturlisten. • ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.	☒
3.	Jeg/vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§14 og 15.	☒
4.	Jeg/vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert i Ephorus, se **Retningslinjer for elektronisk innlevering og publisering av studiepoenggivende studentoppgaver**	☒
5.	Jeg/vi er kjent med at høgskolen vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens studieforskrift §31	☒
6.	**Jeg/vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider**	☒

Figure 1: Self decleration to show that the group has agreed not to cheat

Abstract

Chess is an old and traditional game which has been played the same way for centuries, but the way the game is viewed and analyzed has been fundamentally changed the last few years. Digitizing games of chess today is important for players analyzing games, and for providing the best experience for viewers watching games. However, the methods currently in use utilize expensive electronic chessboards. Aalesunds Schaklag has experienced this exact issue, which is why we decided to help them. The way we did this was by using image analysis to digitize games. Instead of using expensive chessboards, this method only requires a camera and a computer to run a program.

The theory part of the thesis details the theory we initially were planning on using, and also the theory which we switched to using during the project phase. To put it simply, all of the theory is related to the four steps of detecting a chessboard, detecting the squares, detecting the chess pieces and finally tracking the pieces over the course of a game of chess. In the method part of the thesis, the organization of the project is discussed as well as all of the different tools used over the course of the project.

The project has resulted in a program which can detect a chessboard, the chessboard squares and the pieces on them. The project has room for improvement when it comes to the tracking of the game, since we did not get this to work with the desired accuracy before the deadline of the project. Additionally, for different reasons we were not able to test the product at the chess club, however we will provide a guide on how to use it, for the client.

Sammendrag

Sjakk er et gammelt og tradisjonelt spill som har blitt spilt på samme måte i hundrevis av år, men måten sjakk blir sett på og analysert har blitt forandret fundamentalt de siste årene. Digitalisering av sjakk er i dag veldig viktig for spillere som skal analysere sjakkparti og seere som ser på sjakk. Problemet er at de vanligste metodene i bruk, bruker dyre elektroniske sjakkbrett. Aalesunds Schaklag har problemer med dette, som er grunnen til at vi valgte å hjelpe dem ved bruk av bildeanalyse. Istedenfor å bruke dyre sjakkbrett, trenger denne metoden bare et kamera og en datamaskin til å kjøre et program.

Teori delen av oppgaven gir detaljer om teorien vi originalt hadde planlagt å bruke, men også den teorien vi byttet til å bruke underveis. For å si det så enkelt som mulig, så er all teorien relatert til prosessen med å oppdage et sjakkbrett, oppdage feltene på et brett, oppdage sjakkbrikkene og til slutt spore brikkene gjennom et helt parti. I metode delen av oppgaven beskriver vi organisasjonen av prosjektet og de forskjellige verktøyene som ble brukt underveis.

Prosjektet har resultert i et program som kan oppdage et sjakkbrett, feltene på sjakkbrettet og brikkene på feltene. Prosjektet har rom for forbedring når det kommer til sporingen av brikkene, siden vi ikke fikk dette til å virke med den nøyaktigheten vi ønsket. I tillegg fikk vi ikke, for forskjellige grunner, testet programmet på sjakklubben, men vi vil sende instruksjoner om hvordan de kan bruke det.

Glossary

Acronyms

DGT - Digital Game Technology

CNN - Convolutional Neural Network

ROI - Region Of Interest

FEN - Forsyth-Edwards Notation

PGN - Portable Game Notation

NTNU - Norges Teknisk-Naturvitenskapelige Universitet

UML - Unified Modeling Language

API - Application Programming Interface

IDE - Integrated Development Environment

GUI - Graphical User Interface

HTML - HyperText Markup Language

JPEG - Joint Photographic Experts Group

HD - High-Definition

OpenCV - Open Source Computer Vision Library

Contents

Figures

Chapter 1

Introduction

In the introduction we will explain some background information about chess and how it is commonly played today in a digitized world. We will then provide information about our specific project and why the client wants us to complete it. Next we will discuss what the main motivation is for us to complete the project. Then we will discuss what the client expects from a solution and what requirements we have to meet. Finally we will explain how the content of the thesis is organized.

1.1 Background

Chess is a field which has seen significant change and progress in the way it is played and viewed in recent years. For centuries, since the invention of chess, the game was played on a physical chessboards using physical chess pieces. Spectating the game like this could be difficult since the chessboard is quite small, so at tournaments you might have had a dedicated a person spectating the game while manually updating a bigger visual representation of the game. The game can still be played in this fashion; however, the game has been digitized much more in recent years. For example, during the chess world championship, not only is there a camera filming the chessboard, usually there is a digital representation of the chessboard that is updated in real time. Today, the most common way to digitize chess games is to use an electronic Digital Game Technology (DGT) chessboard. According to Digital Game Technology, the main producers of electronic chessboards, electronic DGT chessboards are accurate and fast chess move input devices. The boards register all moves and record the games. They are used for live game transmission, training, online play, computer play and game analysis. Electronic DGT boards are used at World Chess Championships and all major and many minor chess tournaments all over the world since 1998 (Electronic boards, 2021). In addition, the boards are used at home, at chess clubs and chess academies and all kinds of chess events all over the world. The main downside to using a DGT chessboard is that they are quite expensive, coming in at around 7000kr. At the biggest tournaments buying an expensive chessboard can make sense as long as the quality is satisfactory. When playing at home or at smaller chess clubs, however, you might not be able to afford multiple expensive chessboards. The goal of our investigation is to create a program that, by using a camera, should be able to recreate a game of chess digitally. The main purpose of doing this, would be to reduce the total cost of digitizing chess games and analyzing them.

1.2 Motivation for this work

The client we are making the program for is the local chess club, Aalesunds Schaklag. Established in 1913, the chess club is still going strong over a 100 years later. The person from the chess club we were in contact with during the process was the deputy at the club, Arne Unneland, who will be referred to as the contact from here on (Om klubben, 2021). During the process of working on the project we had regular contact with him, to make sure that we were on the same page throughout. At the very first meeting, the contact told us about the chess club and how things are organized there. Among other things, he informed us about the chess boards they use, and that if they want to track games they have to use the expensive DGT chess boards. The main motivation for completing our investigation is to help Aalesunds Schaklag reduce the overall cost of digitizing chess games.

It is also worth noting that image analysis is a big field with many applications, including topics that are directly related to the ones explored in this thesis. Analysing traffic systems or facial recognition are examples of topics that could potentially use similar methods as the ones used in this thesis. This gives us extra motivation when working on the project since the members of the group will gain useful experience working with a relevant topic.

1.3 Requirements

This chapter discusses the requirements that the project should meet in order to call the project a success. It discusses both the financial aspects of the projects as well as the technical requirements. The project should deliver a product which can digitize a game of chess with high accuracy. Some inaccuracy is expected, but there should be ways of correcting these manually. The program should work in a way where two players can sit down and play a game of chess without having to do anything else than play the game itself.

1.3.1 Financial requirements

All of the financial requirements come from the equipment needed. A camera is required, but in theory it does not need to be a camera of very high quality since images of higher quality takes longer to analyze, which is not ideal. A camera capturing at 1080p is probably ideal. To prevent the camera from being in the way of the players, the camera will be mounted to the roof. This means that a way to mount the camera to the roof will also be necessary. In addition to this equipment, a computer that will run the program will be necessary. Finally, a way of connecting the camera to the computer is necessary in order to be able to analyze the video stream.

1.3.2 Technical requirements

As for the technical requirements for the project, we will have to meet four main sub-goals.

- The first problem we will have to solve will be to detect a chessboard from an image. Detecting chessboards or checkerboard patterns are common practice when calibrating cameras, and is also something everyone on the group had experience with from working with the image analysis subject previously.
- Secondly we would have to be able to detect all 64 individual squares. Slightly more challenging, but it should not be more difficult than detecting all the lines on the chessboard and finding their intersections.
- Next, we will need to figure our how to detect all 32 individual chess pieces. This is arguably the most difficult tasks, as the camera will be hanging in the roof, and most chess pieces look very similar from above.
- The final sub-goal will be to track the pieces throughout the game.

Only after completing all of these steps will we be able to track full games of chess and make a program which can support digitization of all chess games.

1.4 Thesis outline

This thesis will further document the work done in connection to the task given by the client. First, the theory part of the thesis will be discussed starting with the research done initially, including the collecting of useful information and looking at existing solutions to similar problems. The theory part will further discuss the other useful theory which was used as a foundation for the solution we ended up using.

After the theory part of the project has been discussed, the methodology will be described, starting with the way the project was organized and which methods were used to communicate. The second half of the methods chapter on the other hand covers the technical tools (programming language, libraries etc.) which were used to develop the program and how they had an effect on the final result.

Once all the background information necessary to understand the final solution has been explained, we will put forward the results from the project. We will give a description of the different components of the solution and why we chose to solve it the way we did.

After the results have been put forward, we will move onto the discussion part of the thesis where the challenges and changes made will be discussed. Most projects encounter obstacles along the way which means there will be changes to the original plans. Our project was no different and we will elaborate on those in the discussion.

Towards the end of the report comes the conclusion, which summarises the results and other experiences made while working on the project. The final part of the report includes the bibliography and the appendix.

Chapter 2

Theory

This is the theory part of the thesis which will inform the reader about the information we researched initially when we approached the assignment. Furthermore, all of the theory necessary to understand the final product will be explained.

2.1 Existing solutions

One of the first choices we had to make, was to decide how we were going to approach the issue. So, we started looking for people who had worked on similar projects. While conducting research we quickly found out that the best way of solving the problem would probably involve the use of convolutional neural networks (CNN). One of the first sources we found that had used a similar method to what we wanted to use with convolutional neural networks was Bakken & Bæck's solution by Saurabh B. and his team (Saurabh, 2019). CNNs are explained in chapter 2.2.

Another existing solution that we took inspiration from was the work done by Andrew Underwood with his project_MYM on GitHub (Underwood, 2020). This solution is similar to the one used by Bakken & Bæck as it detects the board and tiles using image analysis and uses a CNN for determining the different chess pieces on the board, but takes a different approach by using a different model for the CNN.

2.2 Convolutional neural networks

CNNs are a type of deep neural networks that is well suited for analyzing visual imagery. Deep neural networks are a form of artificial neural networks that has multiple layers between the input and output layers. Artificial neural networks, usually just called neural networks are computing systems which vaguely resemble the neural networks in animal brains. Simply put, neural networks use nodes resembling neurons to transmit information to other connected nodes.

CNNs use a process called convolution, which essentially is trying to detect features in regions of images. The features can then be used to classify images into the correct category. The classifying is made through several layers, and the output from those layers. That output is also weighted so that the more defining characteristics are valued higher than more prevalent features. The latter layers usually consists of pooling and fully-connected layers, that are common for most neural networks. Figure 2.1 shows the way convolutional neural networks work using nodes in multiple layers.

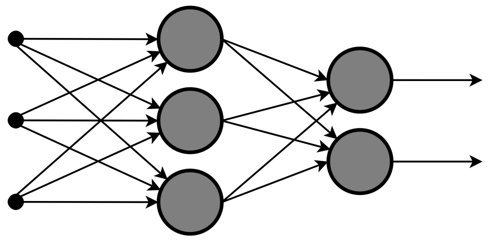


Figure 2.1: A figure showcasing the multiple layers of a CNN(Convolutional neural network, 2021).

2.3 Image analysis

This project relies heavily on the use of image analysis tools and methods. Due to the extensive use of image analysis in this project, the group will research different methods and approaches that best suits the project. We will then utilize the solutions which satisfy the requirements of the client.

2.3.1 Board detection

When a camera captures a chessboard, it is likely that the raw taken image will have less than optimal conditions, since it will capture some of the surrounding area as well. Limiting the analysis to the chessboard only will provide better and more accurate results. Due to this issue, a method or a technique is required that manages to limit the image analysis to only the board and not its surrounding area, such as a table or other surfaces the board may be placed on. The best solution to this issue might be to implement a region of interest (ROI), that tracks the board and crops it from the image for further analysis. It should be possible to detect and crop several boards from the same image with the proper use of different image analysis methods and techniques. Without having to manually selecting the ROI by hand, which will prove to be unpractical as well as time consuming, we may have to use an edge detector to determine the outer edges of the boards as positions to be used in the ROI.

2.3.2 Tile detection

Being able to analyse individual tiles is critical for the overall functioning of the project. If we can separate each tile into their own variable or image, we could more easily run the same analytic process for each individual tile along with the potential chess piece standing on that tile. One way we can achieve this is by detecting the lines at the sides of each tile using an edge detector, and then find the intersecting points of these lines to determine the end-points of each tile. These points may serve as positions / coordinates for each tile to be cropped into individual images.

Edge detection

One of the best ways to detect lines and edges in an image is to use an edge detector. An edge detector manages to determine an edge based on rapid changes in the pixels of an image.

Line detection

We may be able to determine straight lines in an image by using the results of an edge detection. By detecting all of the edges in an image we can detect those that form a straight line together, and use these lines for further processing in tile detection.

2.3.3 Piece detection

The detecting of pieces will mainly be done by a machine-learning method such as a CNN. After taking into consideration what other similar projects have done to overcome their challenges, it is likely that this project will utilize a convolutional neural network in order to correctly determine what type of chess piece is placed on each tile. This will require the use of an available data-set containing images of each piece in different positions, rotations and angles from the camera in order to train the CNN model to achieve a higher accuracy when determining pieces. Upon determining each piece and tile, the result will be saved and used further in the code in the form of FEN and PGN notations More information about FENs and PGNs can be found in chapters 2.6 and 2.7.

2.4 Agile

At the beginning of the project when we were thinking about which development model we were going to use we considered methods like the waterfall model, but ultimately we decided on using agile. It was a fairly easy decision for us, since all of the members of the group had experience working with this model before. Additionally, we knew it would fit our project better since we would be receiving feedback from both the adviser and the client, which would mean we would have to make changes during the project phase, something agile does well. Being agile and adaptable is the main philosophy in agile, so we knew it would fit our project. According to the agile manifesto, the following is the core values of agile development (Manifesto for Agile Software Development, 2001):

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

More specifically, we decided on using Scrum, which will be explained in more detail in chapter 3.6.

2.5 Cohesion and coupling

Cohesion and coupling are two terms which are very often used together in the software industry.

If one class has a wide variety of functionalities, it is said to have low cohesion. This means that the class is broad and not focused on the task it should be doing. If you have high cohesion on the other hand, it means that the class is centered on doing one job instead of multiple. You always want to aim for high cohesion when working on software projects. This way, each component of the project will be more reusable and easier to maintain, since it does not impact the rest of the project as much as if it would have low cohesion.

The other term, coupling, refers to how dependant two classes are on each other. If you have high coupling in your code, it would be difficult to make changes to one part of the code without disrupting something else. If you have low coupling, however, it is much easier to make changes to one class without disrupting another class which is dependant on it. This is why you want to aim for low coupling in your code.

Throughout working on the project we aimed for using high cohesion and low coupling.

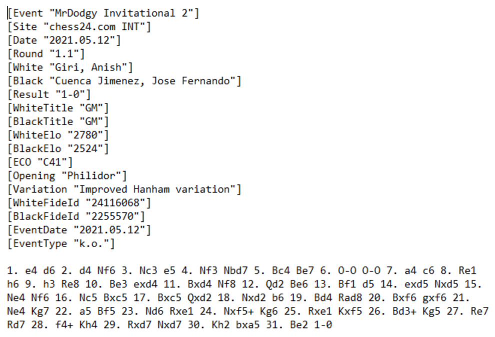
2.6 Forsyth-Edwards Notation

The standard notation for describing a chessboard is Forsyth-Edwards notation (FEN). This notation provides all the necessary information about the current state of the chessboard. The starting position of a chess game using FEN will look like this
rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1
Where the lowercase letters represents black pieces and uppercase letters represent white pieces, the numbers 8 show there are 8 open squares in that row. The lowercase w is to indicate white will do the next move, this will be change to a lowercase b when is it black's move. KQkq shows both castling rights for both

sides, followed by a half-move counter and a full-move counter.

2.7 Portable game notation

Portable game notations (PGN) record the moves that has been made, and the sequence in which they were made. It is made to be a way for computers to easily read a chess match. This format is very suitable for reviewing matches as one can add comments after a particular move on the position, and one can easily use it with a computer to help with the analysis. PGNs also usually stores information about the game such as location, date, who was playing each color pieces, and more as shown in figure 2.2.



```
[Event "MrDodgy Invitational 2"]
[Site "chess24.com INT"]
[Date "2021.05.12"]
[Round "1.1"]
[White "Giri, Anish"]
[Black "Cuenca Jimenez, Jose Fernando"]
[Result "1-0"]
[WhiteTitle "GM"]
[BlackTitle "GM"]
[WhiteElo "2780"]
[BlackElo "2524"]
[ECO "C41"]
[Opening "Philidor"]
[Variation "Improved Hanham variation"]
[WhiteFideId "24116068"]
[BlackFideId "2255570"]
[EventDate "2021.05.12"]
[EventType "k.o."]

1. e4 d6 2. d4 Nf6 3. Nc3 e5 4. Nf3 Nbd7 5. Bc4 Be7 6. O-O O-O 7. a4 c6 8. Re1
h6 9. h3 Re8 10. Be3 exd4 11. Bxd4 Nf8 12. Qd2 Be6 13. Bf1 d5 14. exd5 Nxd5 15.
Ne4 Nf6 16. Nc5 Bxc5 17. Bxc5 Qxd2 18. Nxd2 b6 19. Bd4 Rad8 20. Bxf6 gxf6 21.
Ne4 Kg7 22. a5 Bf5 23. Nd6 Rxe1 24. Nxf5+ Kg6 25. Rxe1 Kxf5 26. Bd3+ Kg5 27. Re7
Rd7 28. f4+ Kh4 29. Rxd7 Nxd7 30. Kh2 bxa5 31. Be2 1-0
```

Figure 2.2: This figure shows what a PGN looks like

Chapter 3

Material and Methods

This chapter discusses how the project is set up. This includes anything from the way the project is organized to which tools were used.

3.1 Organization

The organization of the project is divided into three main parties. This chapter will describe the different parties and which roles they play in the project.

3.1.1 Project group

The project group consists of the three students Tommy Flåm, Jan Olav Skjong and Olav Andreas Grøtta, all of whom are studying at NTNU Ålesund. It is the three of us who make up the contractors of the project and we are the ones who are actually going to complete the investigation. In principle we agreed that everyone in the group should have a shared responsibility to help with everything. We did, however, have some more specific roles including:

- Who should report to the client and the adviser
- Who should write notes during the meetings
- Who is responsible for researching methods which could work better than the methods currently in use

The goal for the project is to have meetings within the project group at least once per week. Ideally, we would have daily meetings, but since most of the members of the group have part time jobs, we would rarely be available for meetings at the same time. The meetings within the project group will take place on Discord. We decided to use Discord since everyone on the group has experience with it from before, so it will be easy to stay in contact this way. We can use Discord for messaging purposes, but also for voice calls or video calls.

3.1.2 Client

The client for this project is the local chess club Aalesunds Schaklag (Sjakk i Ålesund, 2021). It is this chess club which sets the requirements for the final solution. As mentioned in the introduction, the person we were in contact with from Aalesunds Schaklag during the process was the deputy at the club, Arne Unneland. He served as a connection between the client and the project group throughout, and was a contact for the project group.

During the first meeting we had with the contact we discussed several things. One of the first things we discussed was the topic of meetings. We agreed that the project group would be the ones calling in meetings with the contact whenever we felt that we had made significant progress or if we felt that we needed his input on something. Other than that we discussed things related to the requirement of the solution and how he had envisioned the solution. We also came to the understanding that even though it is the client who sets the requirements of the solution, there could be certain obstacles along the way that might slightly change look of the final product compared to how it was initially envisioned. The client had particular sympathy for how difficult it would be to use image analysis to identify each chess piece, when the camera would capture from above, causing each piece to look very similar.

As mentioned previously, in addition to working on the project, some members of the project group have part time jobs as well, meaning that organizing meetings could be difficult. The way we solved this was by first asking the contact to give us a list of times where he was available, and then discussing internally in the project group which of the listed times would fit for all of us. Because of the ongoing COVID-19 outbreak, the plan is for all of the meetings to be held digitally using Zoom video calls.

3.1.3 Adviser

The final party of the three, is the adviser, who has assisted the project group throughout the project. The adviser for the project is Saleh Abdel-Afou Alaliyat, an associate professor at NTNU Ålesund. The role of the adviser is to make sure that there is progress throughout the project and to help the group with problem solving when necessary. Not only is this an advantage for the project group when it comes to completing the project, it also helps the project more closely resemble working life, where it is common for projects to be solved in bigger groups.

For the project we decided to use the agile method Scrum. We decided to use the standard sprint length of two weeks, which meant that it was also natural to have meetings with the adviser at the end of every sprint, every two weeks. The subject of Scrum is explained in more detail in chapter 3.6. The meetings between the project group and the adviser will take place at 10 AM every other Friday, since everyone will mostly be free at this time. The meetings will be held on the communication platform Microsoft Teams. Before the meeting the project group will prepare a status report with various information, such as all the work done during the sprint, what the difficulties were and the goals for the next sprint. This status report was uploaded to Microsoft Teams before every meeting.

3.2 Project planning

Before we started working on the project we had a different subject which covered the topics of project work in general and pre-project planning. One of the mandatory assignments we had in this subject was to develop a pre-project plan, which is attached in the appendix. The plan covers several topics which can be useful to consider before starting work on a project.

At the beginning of the pre-project plan we divided the main roles in the project into three as described in chapter 3.1. Subsequently we agreed on which roles each person in the project group would have. More details about the different roles within the project group can be found in section 3.1.1. We also made certain agreements between the members of the project group, to make sure that everyone worked consistently on the project.

The next part of the plan is the project description, which is largely based on the first meeting we had with the client. Initially when we got the assignment we had a certain idea of what the client wanted from the project, but we did not get the complete description until the first meeting. This chapter of the report also covers topics like how we were planning on collecting information, a risk analysis and how we were planning on progressing through the project.

The subsequent chapter of the pre-project plan covered how meetings would be held, both with the client and with the adviser. It was also agreed upon which actions should take place in case there were any deviations to the plan. The final part of the plan covers which equipment we would need, and how we would acquire it.

3.3 Quality insurance

As mentioned in the previous chapter about the pre-project plan, we made a project description already in the planning phase of the project. This was to ensure that the quality of the project would be insured along the way. As mentioned in section 3.1.1, we have certain responsibilities within the project group to make sure that certain important tasks were always completed throughout the project. Besides the set responsibilities, we have a more fluid system where everyone will help each other if we encounter any obstacles.

For this project we decided to use Scrum, which will be described in more detail in chapter 3.6. One of the most important aspects of using Scrum is that you have regular scrum meetings where all the members of the project group are updated on each others progress. This is an advantage since everyone in the group will have a more complete understanding of the work being done as a whole. At the end of each sprint we would write a progress report containing all work done and what we would work on in the next sprint. Then we would discuss the progress report with the adviser at the end of each sprint. This ensured that we always were on track and did not fall too far behind.

3.4 Criteria for finished task

In the requirement analysis attached in the appendix, we have made a list of requirements that the client expects from the project once it is finished. These requirements were developed throughout the project, but most of them were already set from the first meeting we had with the client.

The requirements include less important ones like the fact that the program should output the time and date along with the PGN of the game. The reason this is not a very important requirement is because there are other formats, such as the FEN format, that can be used to represent the game state. These notations are discussed in more detail in chapters 2.6 and 2.7. The reason that the time and date of the game is not very important is that the only effect is has is that it makes it easier to look up the games afterwards, if you know when you played the game.

On the other side there are some important requirements as well. The more important requirements are mostly centered around the players. That the camera should be mounted to the roof is important because the camera should not be in the way of the players, so that they do not have to worry about blocking the camera with their arms, for example. Additionally, the program should work with a high accuracy, so that the players seldom have to interact with the program to manually correct mistakes.

In addition to these criteria, we were also planning on setting up the system at the chess club and testing it. However, because of the COVID-19 outbreak we are not going to be able to go there in person to set it up. We will most likely solve this by testing the program at home, and then sending the program to the client with instructions on how to set it up.

3.5 Expected documentation

For this project, the following documentation is expected:

- Pre-project plan - The report made before work on the project itself started. Described in more detail in chapter 3.2.
- Requirement analysis - What the client expects from the finished product. Described in more detail in chapter 3.4.
- Unified Modeling Language (UML) diagram - The diagram which shows a visualization of the system can be found in figure 4.1.
- Progress report - The reports written at the end of every sprint, containing information like what was done, what were the challenges and what will be done in the coming sprint.

All of this documentation can be found in the appendix.

3.6 Scrum

During this project we have been using one of the most popular agile frameworks, scrum. Scrum is a framework centered around continuous improvement, and targeted towards teams. Figure 3.1 illustrates the scrum process. There are several platforms which provide Scrum. The platform we decided to use was Jira, provided by Atlassian, since all of the members in the group had worked with Jira before.

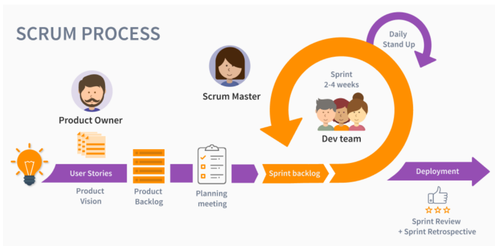


Figure 3.1: A figure illustrating the Scrum process. (Understanding Agile Scrum in 10 minutes, 2021)

3.6.1 Sprint

In scrum there are predefined periods of time knows as sprints, which usually range from between two to four weeks. We decided to use two weeks, since this is the most common nowadays. Some people even use sprints of lower than two weeks recently, since it makes planning easier by making shorter and more focused stories. At the start of every sprint the group agrees what work should be done during the sprint, taking into account how big the group estimates the task is. During the sprint the focus should mainly be on these tasks. It is possible to estimate that you can do more or less than you actually end up doing, in this case you have time to reflect at the end of each sprint by conducting a retrospective.

3.6.2 Meetings

Even though everyone should know what their job is during the sprint, it is common to arrange daily meetings or also called a daily stand-up. This is a short meeting held at the beginning of the day where you tell the other members what you did yesterday and what your focus will be today. Because many of the members of our group were working part time jobs, it was difficult for us to find time to hold daily meetings, but we did hold at least one meeting every week.

3.6.3 Backlog

A backlog in Scrum is an overview of all of the work which has to be done in order to call the project completed. At the start of each sprint the group selects tasks from the backlog which should be completed.

3.6.4 Roles

There are not many roles in Scrum, however you need someone with responsibility. Here we will mention the most common roles.

Product owner

The product owner is the person representing the owner of the product. The product owner decides which functionalities the product should have, and what the project group should focus on. In our case the product owner is Arne Unneland, representing Aalesunds Schaklag.

Scrum Master

Usually there is no boss in Scrum, however there still is a person who gets the title Scrum Master. This is the person who has the responsibility of making sure that everything is progressing as it should. The Scrum Master should also manage the backlog and make sure that all the deadlines are met.

3.7 Libraries

When working on the project we used a few external libraries to expand on the functionalities of Python. Some of the libraries were used for machine learning, others for general image analysis and so on. This chapter will discuss which libraries were used and why we decided to use them. It will also shortly be mentioned some examples of how we used them.

3.7.1 TensorFlow

In the very beginning when we started researching the project, we found that the best way to solve the project would involve using machine learning. After conducting some further research we found that the best way to do this would probably involve the use of TensorFlow. TensorFlow is one of the most popular open-source software libraries in the field of machine learning. It is widely used by many of the biggest technology companies in the world, like Google or Intel, as well as other companies like Airbnb or Coca-Cola (Why TensorFlow, 2021). TensorFlow was initially developed by the Google Brain team for internal use at Google only. However, in 2015 it was released to the public for others to use. The TensorFlow library is made to be able to train models as fast as possible by using

Tensors. Tensors are mathematical arrays consisting of multiple arrays or variables (Introduction to Tensors, 2021). We used the latest stable release, which at the time was 2.4.1.

3.7.2 Keras

The next library which we used is Keras. Keras is a library which is very common to use together with TensorFlow, as it serves as an interface for TensorFlow. According to Keras themselves, Keras is an application programming interface (API) designed for human beings, not machines. The purpose of Keras is to offer consistency and simplicity in order to reduce the number of user actions required when working with deep learning and TensorFlow in particular. This is also the case for debugging, as Keras aims to provide clear and actionable feedback upon error detection (Keras, 2021). We have used version 2.4.0 of Keras, which was the latest stable release at the time.

3.7.3 OpenCV

The third library we used when working on this project is perhaps the most important one. OpenCV stands for Open Source Computer Vision Library and is an essential library for anyone doing work within the field of image analysis. Weather you are working with Python, Java or MATLAB, OpenCV is likely going to be a must have. According to OpenCV, they have over 2500 optimized algorithms, including both classic and state-of-the-art computer vision and machine learning algorithms (OpenCV, 2021). We used OpenCV for multiple functions including the Harris corner detector and the Hough line detector, to mention some.

3.7.4 Chess

Another library that we used was the standard Python chess library. This library provides general chess features such as move generation and move validation. It also provides functions that can detect checkmates and stalemates, for example (python-chess: a chess library for Python, 2021). One of the functionalities of the chess library we used is the generation of FENs, in order to compare the board state before and after a move.

3.7.5 Chessnut

This library is similar to the Chess library. It can import and export FENs, as well as list the legal moves from a board state and apply moves. We have chosen to use Chess over Chessnut for everything except finding the legal moves from a board state, as Chess does not have this function.

3.8 Tools

To be able to solve the task given to us by the client, we had to select a programming language to write a program and also an integrated development environment (IDE) to write the program in. This chapter will describe these tools, why they were chosen and how they were implemented.

3.8.1 Python

After deciding to use TensorFlow, we had to decide which programming language we were going to use. At the start of the project we did not have much experience working with Python. However, we quickly decided that the best solution would be to use Python. Considering the fact that we had already decided early on that we were going to use TensorFlow, it was only natural to use Python since this is the preferred language to use with TensorFlow. It is not necessary to use Python with TensorFlow, as it is totally possible to use other programming languages, like C++ or JavaScript for example. However, TensorFlow was initially made for Python and it is also the most popular option. Not only is Python a popular option with the TensorFlow library, Python has become one of the biggest programming languages in the world in recent years with many existing libraries with functionalities potentially useful to us. This was indeed the most important aspect for us when choosing this particular language. This meant that for the first couple of weeks we would have to learn how Python works before we could get started working on the project. As a group we decided that it would be worth it considering the fact that the programming language is quite popular these days, so learning it would likely benefit us in the future anyway. We used Python version 3.9.2, which was the latest stable release when we started working on the project.

3.8.2 Spyder

After deciding on which programming language to use, and which libraries we were going to use, the next step was deciding which IDE we wanted to use. Although it is not necessary for everyone in the group to work with the same IDE, we decided to use the same one, as we thought it could possibly simplify collaboration. When working with Python you have quite a few options, however we decided not to think too much about it as it should not have too big of an effect on the final product. Since everyone on the group had worked a bit with Anaconda before, we all had Spyder installed from before, which made the choice easy for us. Initially we used different versions of Spyder, however when version 5.0.0 came out we upgraded to this version.

3.9 Technologies

In this part of the thesis we will describe various technologies used when working on the project and how these helped us when working on the project.

3.9.1 Git

When you are working on the same project together you need to be able to effectively share the work which is being done. To solve this, we have chosen the same solution as so many others, which is to use Git. This was not a difficult choice seeing as all of the members of the group had experience working with Git before. Git is a free and open-source version control system which can be used when working on projects whether they are small or very large (Git, 2021). When using Git, changes to the code can be published through a commit, containing a description of which changes have been made. This also makes it possible to go back to previous solutions, should something go wrong with a new version of the code. There are several other options to Git, but considering the fact that Git is by far the most popular, and as mentioned earlier everyone on the group had experience working with it, we saw no reason to suddenly use something new. The most recent version of Git we used was version 2.31.1.

3.9.2 Sourcetree

Similarly to Git, everyone on the group also had experience working with Sourcetree. Sourcetree is a graphical user interface (GUI) made by Atlassian which makes using Git easier. It provides a visual representation of the repositories you are working with, which simplifies everything. This in turn allows the user to spend more time coding instead of working with the Git command line (Sourcetree, 2021). There are countless competitors to Sourcetree, but seeing as everyone on the group had worked with Sourcetree before we saw no reason to switch now all of a sudden, especially considering the fact that Sourcetree is one of the most popular choices.

3.9.3 Confluence

In chapter 3.6 we discussed the use of scrum and Jira. Jira is perhaps the most popular product developed by Atlassian, but they also have a product called Confluence. Confluence was initially released in 2004, but it is still commonly used today (Confluence, 2021). When working on software projects in a group it is usually very useful to have a workspace to collaborate. Confluence provides a workspace with a plethora of functionalities. The functionality we probably will be using the most for this project is the Gliffy diagram creator. Gliffy is a software used to create a wide variety of diagrams online, via an HTML5 cloud-based app (Gliffy, 2021). It is very easy to use, and is very convenient when making diagrams together. We will at the very least be using Confluence to create activity diagrams and class diagrams for our project. Other common uses for Confluence include making retrospectives and meeting notes, for example. For this project we will be using Microsoft Word for these purposes, however, as they simply better fit our needs better.

3.9.4 Overleaf

When we started writing on the thesis we had to decide on which editor we were going to use. We discussed it with the adviser and he recommended that we use Overleaf. Overleaf is is a collaborative cloud-based LaTeX editor aimed at people writing scientific documents (Overleaf, 2021). LaTeX is a free software system for document preparation where the user uses markup tagging to format the document in the way they want (LaTeX: a document preparation system, 2021). Initially we found Overleaf to be a bit overwhelming, but as we got more experience working with it, we had a very good time working with the editor.

3.10 Data

The collection of data for the project happened in a variety of ways. Most of the data we used when working on the project comes in the form of images. When working with images like we did it is usually a good idea to use the same format for all images. In this case we decided to use JPEG since this is the default format for the cameras we used. We could have used different formats but we did not see a reason since we got fine results with JPEG.

Initially, the plan was to visit the chess club and take the pictures there, to better get an impression of how the final result would be used. Unfortunately, because of the Covid-19 outbreak we did not get the chance to visit the chess club. However, the client was able to go to the chess club to take pictures for us. We got a set of pictures of one chessboard per picture. However, in the final setup the plan is for the camera to film two chessboards at once, so we got a set of pictures resembling this as well. These pictures were going to be the ones resembling the final setup the most, so it was really helpful to have them.

In addition to the pictures from the chess club, we took some more pictures of different chessboards to test how the program would perform with different inputs. Even though all chessboards have the nature of having 64 squares alternating between a black color and a white color, chessboards can wary. This is particularly true when it comes to the colors, since some chessboards have a very dark black color and a very bright white color while other chessboards can have a more brown black and a beige white. The difference in color is illustrated in figures 3.2 and 3.3. We also took pictures from different heights, to see if the program could handle this. These are the reasons for why we decided to gather data in a few different ways and not just the pictures from the chess club.

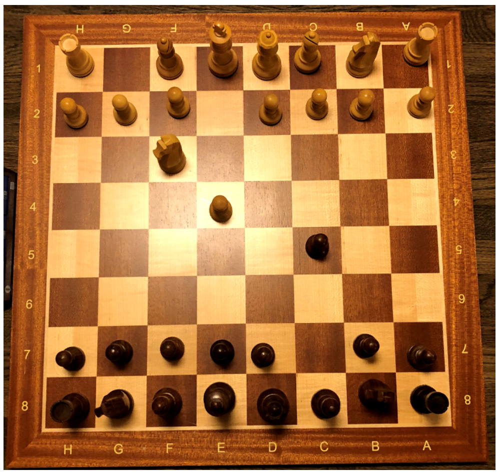


Figure 3.2: What a brown and beige chessboard might look like.

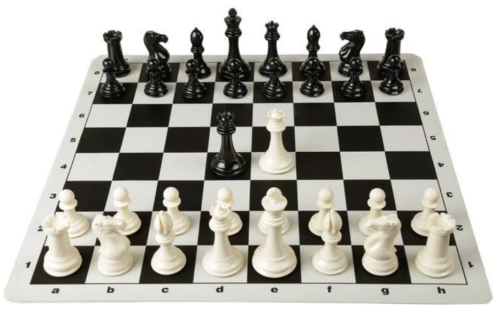


Figure 3.3: What a black and white chessboard might look like.

3.11 Equipment

This chapter describes which physical equipment was needed when conducting work on the project. The equipment was financed by the client.

3.11.1 Camera

Perhaps the most important piece of equipment required for this project, is the camera used for capturing the chessboard. As previously mentioned in the introduction, the quality of the camera should not need to be too good, as it will take longer to process. However, the quality needs to be good enough so that each square and the pieces on them can be seen clearly. The camera we will be using will capture at 1080p, which fits the requirement of having good quality, but not anything too extreme. The camera will be mounted in the roof approximately one and a half meter above the chessboard, so as long as the camera can capture good quality from there it will be satisfactory.

The client decided to buy a Logitech HD Pro Webcam, a camera which can be used as a webcam, but also it can be mounted to a tripod and attached to the roof using a double sided tape type mechanic. The fact that it can be used for multiple purposes can be of particular advantage at the moment, since the chess club is not open currently because of the ongoing COVID-19 outbreak. This means that the camera can be used as a webcam for the moment, and then when the chess club opens again it can be used to capture the games. Figure 3.4 shows what the camera looks like.

Figure 3.4: Logitech HD Pro Webcam C920

3.11.2 Computer

Another important device for the project is a computer. The computer is necessary because the camera needs to be connected to a computer which can run the program. Decent hardware should suffice to run the program. Figure 3.5 shows the computer we will use for our project.

Figure 3.5: Khameleon P9 Laptop Gaming PC

3.11.3 Chess set

The final piece of equipment needed for the project is perhaps the most obvious one, a chess set. A chess set includes a standard chessboard with 64 squares preferably labeled from A to H and 1 to 8. The other part of a chess set is the chess pieces which include 16 pawns, four rooks, four bishops, four knights, two kings and two queens. The total number of chess pieces should add up to 32, 16 for each player. In addition to the chessboard and the chess pieces, many players also use a clock with a timer for each player. If you play casually however, many players opt out of buying a clock. Figure 3.6 shows the chess set we will use for our project, which is also a good representation of what a typical chess set might look like.

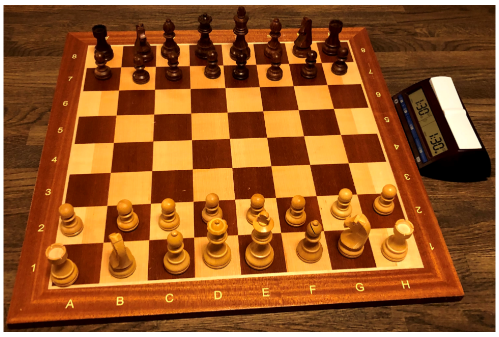


Figure 3.6: What the chess set looks like at the chess club.

Chapter 4

Results

This chapter provides information on the different results we have achieved throughout the project. Information regarding the system as a whole will be shown first, before we provide some insight on how the different parts of the project has been developed.

4.1 The system as a whole

When the system is put together one can see how the workings of the project goes. After an input image from the web-camera comes through, the system will read the image and proceed with making it easier to work with by converting the image into gray-scale and applying a blur effect. This removes most unwanted noises from the image and makes the detection of contours easier. The largest contours, the board itself, is saved as a ROI. The system then detects each line in the ROI and separates them into vertical and horizontal ones before finding the crossing point between them. The points are enhanced to serve as positions for every tile on the board, which are then cropped and saved. Each saved tile is processed to detect if there is a piece standing on them and what piece it is. The system then tracks the movement of the pieces and gives an output of the positioning of every tile on the board in the form of a PGN file along with the date and time of the chess game. Figure 4.1 illustrates how the system works.

4.2 Image analytic process

The image analytic process is the core aspect of the project, as such, major parts of the work is centered around the idea of processing changing and augmenting images.

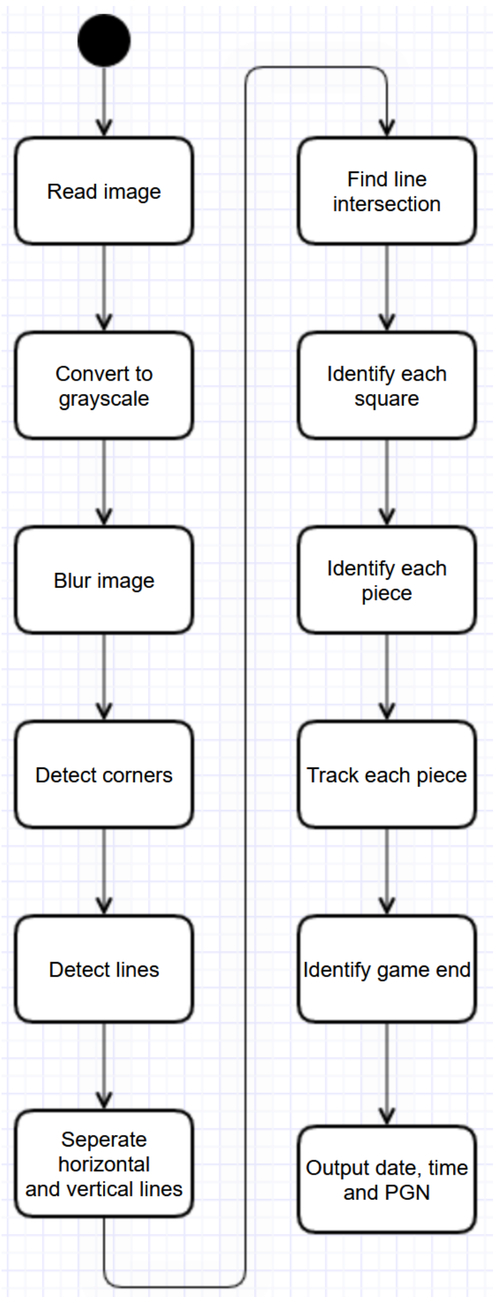


Figure 4.1: An activity diagram showing the steps of the system.

4.3 The process of chessboard detection

The original plan was to process several chessboards from the same image, but due to inconsistencies and inaccuracies with more then one board this was changed to only focusing on a single board.

4.3.1 Low level OpenCV techniques

The use of OpenCV during the analytic process was essential for detecting and focusing on the chessboard from the raw images. After acquiring the input image, the process starts with re-scaling the image to a smaller frame for easier work within the code, as having too large of dimensions on the image could cause inconsistencies or inaccuracies with certain outputs further later in the code. The re-scaling of images, among other functions are taken from an already existing set of function found on GitHub, project_MYM by Andrew Underwood. The re-scaling function as well as other functions are found in their *cv_chess_functions.py* file and is used throughout the entire image analytic process. Figure 4.2 show a chessboard after being re-scaled.

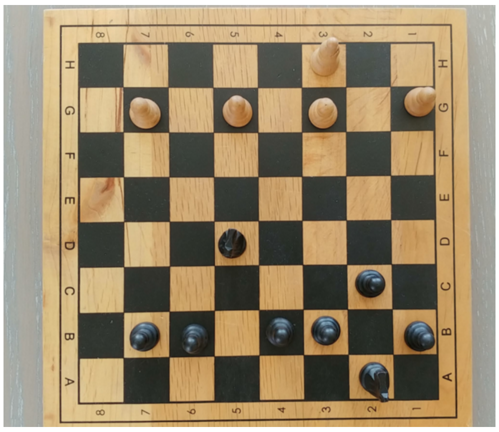


Figure 4.2: Re-scaled chessboard

Upon re-scaling, the process continues by turning the image gray-scale and blurring it by using OpenCV's cvtColor() function with the COLOR_BGR2GRAY as a parameter, and their blur() function to reduce possible noise in the image. Figure 4.3 shows the effect these functions have applied to the image while figure 4.4 show the code used.

4.3.2 Edge detector and contours

The processed image is then put through OpenCV's canny() function, an edge detector, where the lower threshold is set to 100 and the upper to 200 for estimating the edges in the image. Figure 4.5 shows the result of the canny edge detection,

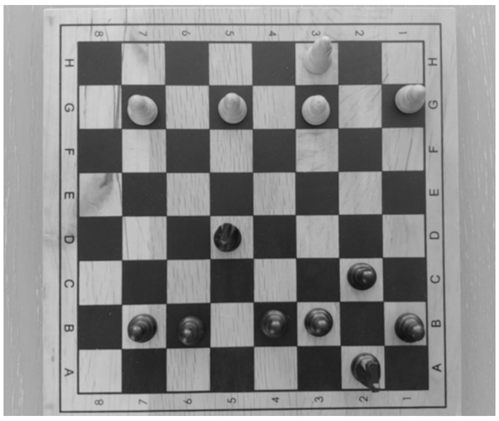
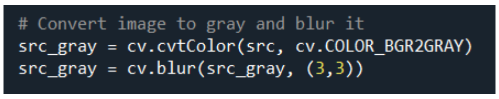


Figure 4.3: Chessboard with gray-scale and blurring applied



```
# Convert image to gray and blur it
src_gray = cv.cvtColor(src, cv.COLOR_BGR2GRAY)
src_gray = cv.blur(src_gray, (3,3))
```

Figure 4.4: The code that applies the effects

while figure 4.6 shows the code for both canny detection the code for detection of contours.

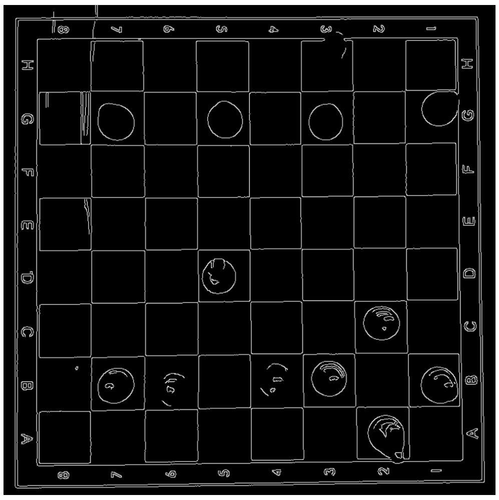
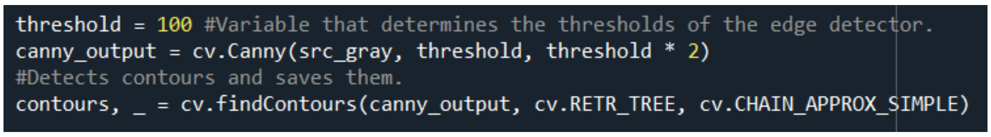


Figure 4.5: The result of canny edge detection

The result of the edge detection is then used in OpenCV's findContours() function to determine all of the contours in the image by utilizing RETR_TREE in the first parameter which retrieves all the contours and sets up a hierarchy of nested contours, as well as CHAIN_APPROX_SIMPLE in the second parameter which only saves the end points of each contour for faster processing times.



```
threshold = 100 #Variable that determines the thresholds of the edge detector.
canny_output = cv.Canny(src_gray, threshold, threshold * 2)
#Detects contours and saves them.
contours, _ = cv.findContours(canny_output, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)
```

Figure 4.6: The code that applies edge detection and the detection of contours

4.3.3 Cropping the board

Once the contours have been detected, we can then create an array to place them in, and sort them by size. This will result with the first contour in the array being the largest one, which will be the chess board itself. By using this contour in OpenCV's boundingRect() function we can assign 4 variables to determine the rectangular position of the board. Continuing from this, we can create a ROI by using the coordinates from the boundingRect() function on the down-scaled image before the gray-scaling and blurring was applied. We have now narrowed down the area to the board only in which further image processing will take place. Figure 4.7 shows the ROI of the board and figure 4.8 displays the code used.

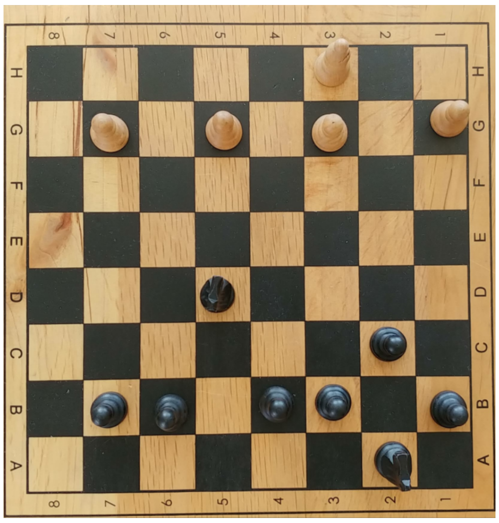
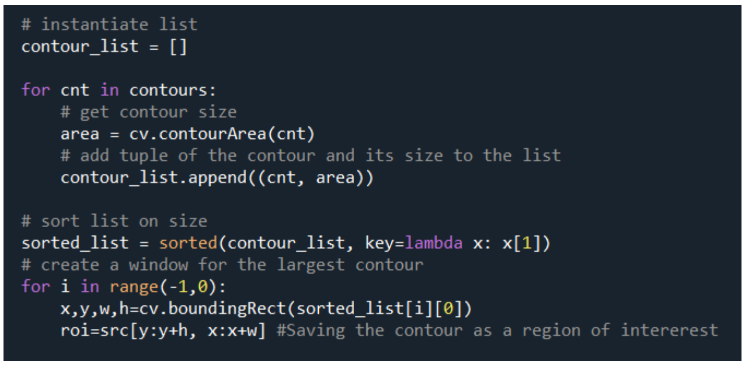


Figure 4.7: The cropped board.

4.4 The process of tile detection

Tile detection is achieved by cropping out each individual tile on the board. After the ROI has been established as its own variable, we can then proceed to use project_MYM's pre-made functions to cleanly implement further processing techniques and functions. The first 2 functions from project_MYM, read_img() and canny_edge(), are more or less the same as the ones described in the process of board detection as they apply gray-scale, blurring and canny edge detection to



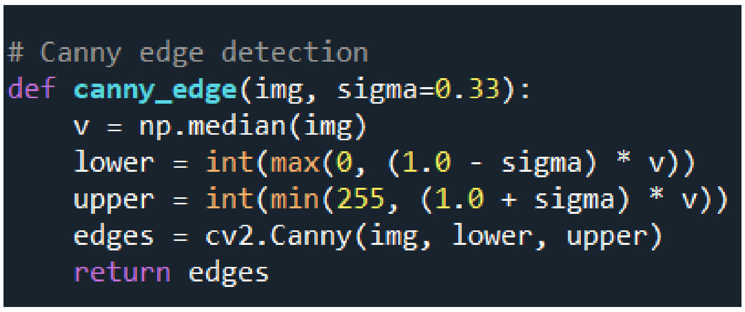
```python
# instantiate list
contour_list = []

for cnt in contours:
    # get contour size
    area = cv.contourArea(cnt)
    # add tuple of the contour and its size to the list
    contour_list.append((cnt, area))

# sort list on size
sorted_list = sorted(contour_list, key=lambda x: x[1])
# create a window for the largest contour
for i in range(-1,0):
    x,y,w,h=cv.boundingRect(sorted_list[i][0])
    roi=src[y:y+h, x:x+w] #Saving the contour as a region of intererest
```

Figure 4.8: Code for cropping the board.

the ROI by using similar parameters. The main difference between our code and theirs is that their edge detector is a bit more fine tuned. Where the board's detection process was only meant to focus on finding the largest contour, the tile's edge detection additionally uses the median of the ROI as a variable for determining the minimum and maximum thresholds. Figure 4.9 displays the edge detection code used for the tiles.



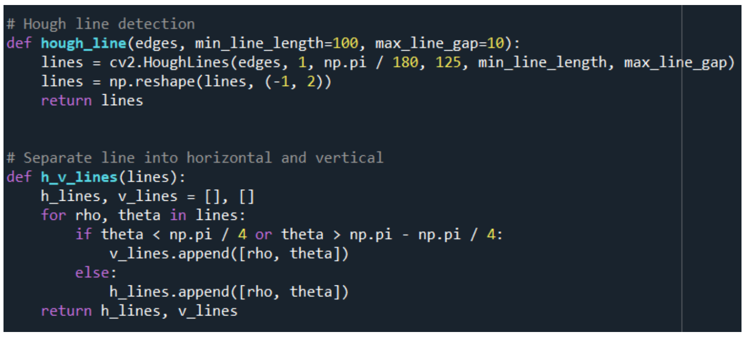
```python
# Canny edge detection
def canny_edge(img, sigma=0.33):
    v = np.median(img)
    lower = int(max(0, (1.0 - sigma) * v))
    upper = int(min(255, (1.0 + sigma) * v))
    edges = cv2.Canny(img, lower, upper)
    return edges
```

Figure 4.9: Code for edge detection on tiles.

4.4.1 Tile lines and points

After processing the image through project_MYM canny edge detector, the image is then processed through their hough_line() function. This function utilizes OpenCV's HoughLines() function, with the canny edge detection as one of the parameters, and numpy's reshape() function as a second parameter. Numpy's reshape() function allows the changes of shape to the arrays of lines without changing any data. Continuing on, their h_v_lines() function manages to separate each line into 2 different arrays, one for horizontal lines and one for vertical lines. Figure 4.10 displays the code that detects and categorizes the lines.

By using the resulting outputs from the their h_v_lines() function, we were able to determine the position in which horizontal and vertical lines crosses over each other as intersection points. These points are then processed in the clus-

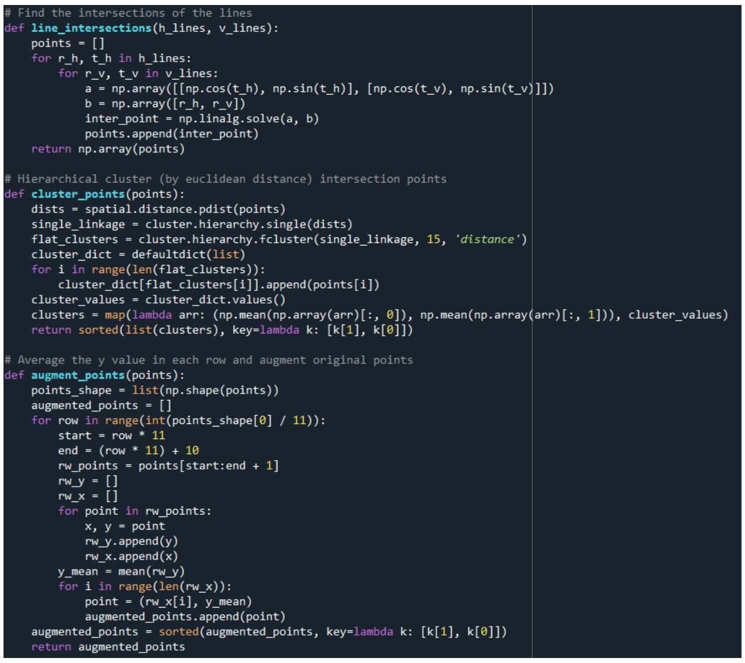


```python
# Hough line detection
def hough_line(edges, min_line_length=100, max_line_gap=10):
    lines = cv2.HoughLines(edges, 1, np.pi / 180, 125, min_line_length, max_line_gap)
    lines = np.reshape(lines, (-1, 2))
    return lines


# Separate line into horizontal and vertical
def h_v_lines(lines):
    h_lines, v_lines = [], []
    for rho, theta in lines:
        if theta < np.pi / 4 or theta > np.pi - np.pi / 4:
            v_lines.append([rho, theta])
        else:
            h_lines.append([rho, theta])
    return h_lines, v_lines
```

Figure 4.10: Code for detecting and categorizing lines.

ter_points() function which manages to set a minimum distance between each point which prevents incomplete tiles from being cropped. These point are finally processed in the augment_points() function which averages the y value for each row and augments the original points. Figure 4.11 displays the different functions that is used for determining the points.



```python
# Find the intersections of the lines
def line_intersections(h_lines, v_lines):
    points = []
    for r_h, t_h in h_lines:
        for r_v, t_v in v_lines:
            a = np.array([[np.cos(t_h), np.sin(t_h)], [np.cos(t_v), np.sin(t_v)]])
            b = np.array([r_h, r_v])
            inter_point = np.linalg.solve(a, b)
            points.append(inter_point)
    return np.array(points)

# Hierarchical cluster (by euclidean distance) intersection points
def cluster_points(points):
    dists = spatial.distance.pdist(points)
    single_linkage = cluster.hierarchy.single(dists)
    flat_clusters = cluster.hierarchy.fcluster(single_linkage, 15, 'distance')
    cluster_dict = defaultdict(list)
    for i in range(len(flat_clusters)):
        cluster_dict[flat_clusters[i]].append(points[i])
    cluster_values = cluster_dict.values()
    clusters = map(lambda arr: (np.mean(np.array(arr)[:, 0]), np.mean(np.array(arr)[:, 1])), cluster_values)
    return sorted(list(clusters), key=lambda k: [k[1], k[0]])

# Average the y value in each row and augment original points
def augment_points(points):
    points_shape = list(np.shape(points))
    augmented_points = []
    for row in range(int(points_shape[0] / 11)):
        start = row * 11
        end = (row * 11) + 10
        rw_points = points[start:end + 1]
        rw_y = []
        rw_x = []
        for point in rw_points:
            x, y = point
            rw_y.append(y)
            rw_x.append(x)
        y_mean = mean(rw_y)
        for i in range(len(rw_x)):
            point = (rw_x[i], y_mean)
            augmented_points.append(point)
    augmented_points = sorted(augmented_points, key=lambda k: [k[1], k[0]])
    return augmented_points
```

Figure 4.11: Code for determining point.

4.4.2 Cropping each tile

With the final points and their position in place the actual cropping of each tile can finally begin. Project_MYM's pre-made function write_crop_images() utilizes 4 parameters where 2 of them are the output from the board detection, the ROI, as well as the output points from the augment_points() function. The third parameter is a simple counter for how many tiles has been processed, and the forth parameter describes the folder-path in which the cropped tiles will be stored. The function manages to create a start_x and start_y position as well as a end_x and end_y position of each tile by utilizing the nearest points in the desired direction. These positions are used to crop out one tile individually starting from the tile on the lowest row and in the rightmost column. This section of the function will continue to run for each tile in a row going from right to left, and start over on the next row until each tile in every row has been cropped and saved within the defined folder. Figure 4.12 displays the result of cropping a single tile while figure 4.13 shows the code that is used to achieve this.

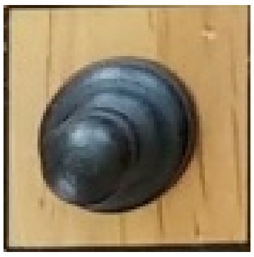
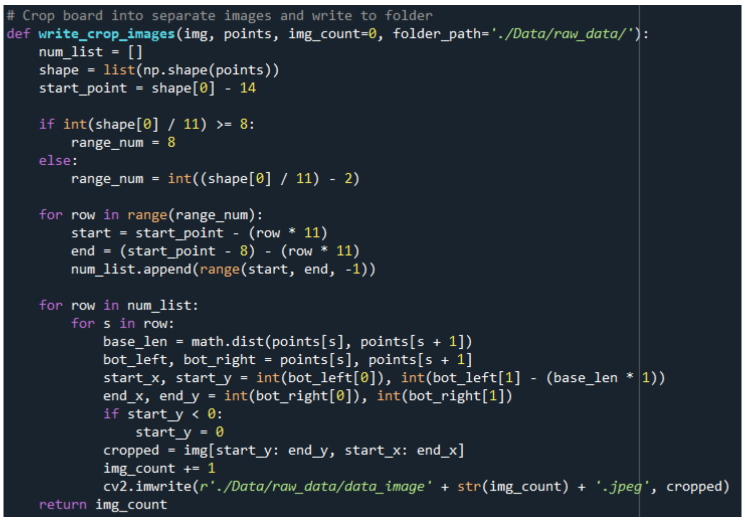


Figure 4.12: A cropped tile from the chess board.



```python
# Crop board into separate images and write to folder
def write_crop_images(img, points, img_count=0, folder_path='./Data/raw_data/'):
    num_list = []
    shape = list(np.shape(points))
    start_point = shape[0] - 14

    if int(shape[0] / 11) >= 8:
        range_num = 8
    else:
        range_num = int((shape[0] / 11) - 2)

    for row in range(range_num):
        start = start_point - (row * 11)
        end = (start_point - 8) - (row * 11)
        num_list.append(range(start, end, -1))

    for row in num_list:
        for s in row:
            base_len = math.dist(points[s], points[s + 1])
            bot_left, bot_right = points[s], points[s + 1]
            start_x, start_y = int(bot_left[0]), int(bot_left[1] - (base_len * 1))
            end_x, end_y = int(bot_right[0]), int(bot_right[1])
            if start_y < 0:
                start_y = 0
            cropped = img[start_y: end_y, start_x: end_x]
            img_count += 1
            cv2.imwrite(r'./Data/raw_data/data_image' + str(img_count) + '.jpeg', cropped)
    return img_count
```

Figure 4.13: Code for cropping tiles.

4.5 The process of piece detection

Piece detection is done by analyzing each of the cropped images of tiles, as shown in Figure 4.12. And transforming it into a binary image. We then proceed to find the largest contour in the image, and measure the size of that. If there is a piece in the tile this method will detect a piece, however it will not tell us what kind of piece this is as shown in Figure 4.15.

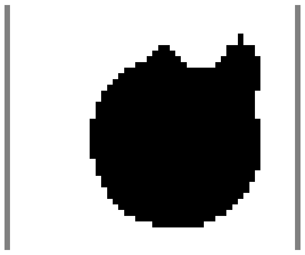
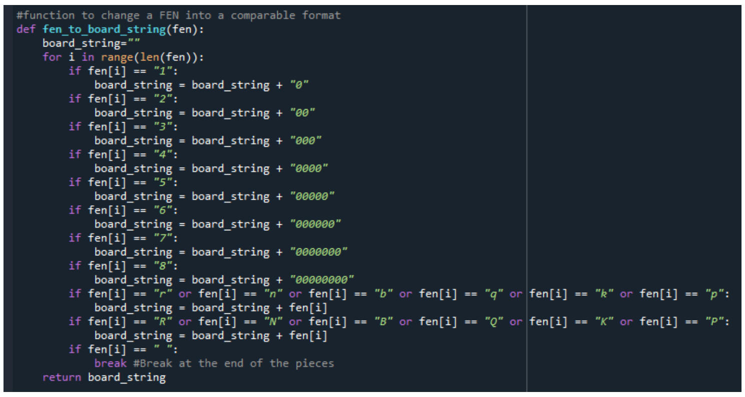


Figure 4.14: An example of a binary image of a tile with a piece in it.

When this is done for each tile on the board we get a string showing the location of pieces. An example of such a string looks like this:
11111111111111110000000000000000000000001000000000001111011111111111

As we only need to find the differing squares from the board after a move and before the move to find which pieces have been moved and we have the board represented as a FEN string we needed to reformat that into a more comparable string format. To do this we simply replaced all the number values in the FEN string with 0 to represent each empty square. So the FEN string of the starting position looking like this:
rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1
Will be reformatted into this:
rnbqkbnrpppppppp00000000000000000000000000000000PPPPPPPPRNBQKBNR

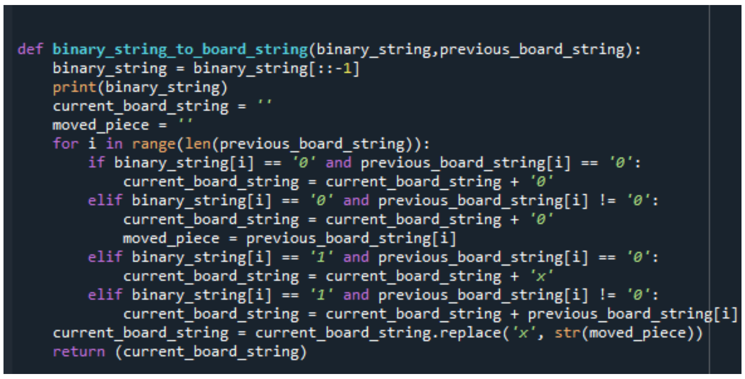


```
#function to change a FEN into a comparable format
def fen_to_board_string(fen):
    board_string=""
    for i in range(len(fen)):
        if fen[i] == "1":
            board_string = board_string + "0"
        if fen[i] == "2":
            board_string = board_string + "00"
        if fen[i] == "3":
            board_string = board_string + "000"
        if fen[i] == "4":
            board_string = board_string + "0000"
        if fen[i] == "5":
            board_string = board_string + "00000"
        if fen[i] == "6":
            board_string = board_string + "000000"
        if fen[i] == "7":
            board_string = board_string + "0000000"
        if fen[i] == "8":
            board_string = board_string + "00000000"
        if fen[i] == "r" or fen[i] == "n" or fen[i] == "b" or fen[i] == "q" or fen[i] == "k" or fen[i] == "p":
            board_string = board_string + fen[i]
        if fen[i] == "R" or fen[i] == "N" or fen[i] == "B" or fen[i] == "Q" or fen[i] == "K" or fen[i] == "P":
            board_string = board_string + fen[i]
        if fen[i] == " ":
            break #Break at the end of the pieces
    return board_string
```

Figure 4.15: The Code for reformatting FEN strings.

4.6 Move detection

By using a reformatted FEN, as shown in chapter 4.5, and the start position through a FEN we keep track of the pieces position for each move made. We match each number in this string against the corresponding index in the reformatted FEN before the move was made. From that we can create a new reformatted FEN.



```python
def binary_string_to_board_string(binary_string,previous_board_string):
    binary_string = binary_string[::-1]
    print(binary_string)
    current_board_string = ''
    moved_piece = ''
    for i in range(len(previous_board_string)):
        if binary_string[i] == '0' and previous_board_string[i] == '0':
            current_board_string = current_board_string + '0'
        elif binary_string[i] == '0' and previous_board_string[i] != '0':
            current_board_string = current_board_string + '0'
            moved_piece = previous_board_string[i]
        elif binary_string[i] == '1' and previous_board_string[i] == '0':
            current_board_string = current_board_string + 'x'
        elif binary_string[i] == '1' and previous_board_string[i] != '0':
            current_board_string = current_board_string + previous_board_string[i]
    current_board_string = current_board_string.replace('x', str(moved_piece))
    return (current_board_string)
```

Figure 4.16: The Code for matching a binary strings to a reformatted FEN.

This will allow us to set each 1 to the corresponding letter in the reformatted FEN, and add up the 0 to format it back into a FEN. Thereby we only need to change the piece that has been moved, which will always be in the reformatted FEN index that matches 0, but the reformatted FEN has a piece there. We can then see the legal moves that this piece can make through using Chessnut's legal move function and test each move against our new reformatted FEN until we find one that matches. That will allow us to get the correct move, and we can push this on a Chess.Board() to make the move. This is done to easily obtain a new FEN and correctly write the move to the PGN file.

Chapter 5

Discussion

This chapter will focus on different perspectives of the work we have done throughout the project. First we will discuss our thoughts on the result part of the thesis, before moving on to the different methods we used and what our experiences were with them. A common theme throughout the discussion is that we were limited in some ways because of the ongoing COVID-19 situation.

5.1 Image analysis

In this section we will discuss the different approaches with the usage of image analysis.

5.1.1 Chess board detection

During the start of the project we had the idea of detecting several chessboards from the same image. This idea led us to create the code of the project in such a manner so that it would support the ability to detect several boards from the input image at once. After conducting some research on the subject, we found out that the usage of several ROI would be effective at detecting several boards in the input image. After researching and finding a working concept of multiple ROI from a single image, we started to take inspiration from it and base our code on this for implementation in this project. Even though we were able to create multiple ROI from an input image, we ran into the issue of there being too many possible ROI in our test images, which would mean that the images taken at the chess club would also suffer the same result. With this in mind we decided to find another solution to better base our work on.

This lead us to continue researching and finding potential solutions to our issues. After finding a promising technique, we managed to utilize a code we found which achieved the same result of detecting multiple objects in the image but also manages to order the largest objects in the image based on their size and work on those and only those (J.D, 2019). With this solution we were able to set up the detection of several boards from a single image.

If the input images are of high dimensions, the image may not be fully processed and result in a incomplete image. It is also worth noting that if the dimensions of the input image are lower than 1080p, the quality of the image and its features may not be detected properly which may result in issues detecting all the tiles on the board.

Later throughout the project it was decided that there should only be one board in the input image. This decision was made due to the results from using more than one board could provide inaccuracies, and missing points regarding the cropping of certain positions on the board. In a optimal scenario, both of the boards would be perpendicular towards the web-camera mounted in the ceiling, which would allow for minimal deviation of the prediction-making process for the clustered points function within the tile detector. Knowing that these boards are not mounted to the table, and are meant to be moved around, we came to the conclusion that if the code was able to detect 2 boards, and one of the boards had a too rough of an angle compared to the web-camera, the resulting output for that board would not be of beneficial value as the output would include other areas then the intended one. It could be possible to solve this issue using pre-processing in the future, but we did not have the time to explore this further. Another option is having two cameras, one for each chessboard, and then having two instances of the program running in parallel, one for each chessboard.

5.1.2 Tile detection

The use of tile detection began with the creation of a simplified code compared to the one used in the results. This code consisted of 2 functions, detectCorners(), and detectLines(). The function detectCorners() utilized numpy's float32 function as well as OpenCV's cornerHarris() function, a different method for corner detection. The function detectLines() utilized most of the methods found in the final version of the code, as it used OpenCV's canny() edge detection function and their houghLines() function. This version managed to create lines along the edges of each tile, but work on this version got halted as we made little progress and when we found project_MYM's code which did the same thing but more, we decided to forgo this early version for the benefit of their code.

The condition of the surrounding area of the chessboard may affect the resulting output. The lighting of the room and area can reflect light from the board onto the camera, and may affect the programs ability to detect some tiles. It should also be noted that a board with a angle too different from the web-camera may result in issues with tile detection. The board should be at a perpendicular angle compared to the board to prevent the resulting output images from being displayed as incorrect areas instead of the intended tile.

From the test images we had available during development we were able to test and exclude certain approaches. Since the chessboards used by the client can reflect a decent amount of light from a source of light onto the web-camera, it became a bit of a challenge to detect certain tiles as the lighting proved to be too much for some methods working on threshold. One of these methods was the otsu method for automatically detecting the thresholds of the image, as it differentiated the area where light reflected onto the web-camera from rest of the board resulting in a large portion of the tiles were not detected.

Due to the test images we were also able to determine the optimal parameter values for some of project_MYM's functions. One of these functions was their write_crop_images() function, that handled the cropping and saving of individual tiles. When we tested their code with our test images, we discovered that the resulting images would show the intended area of the crop, but also half the tile above. This made us change the code by lowering the calculation for the start_y variable by simply changing the code: (base_len * 2) into (base_len * 1).

5.2 Piece detection

The plan was to use CNN as image recognition to classify the pieces. We did a lot of testing on this, and tried to set up several different models through Keras, but we did not get this working. Eventually as we got pressed on time we decided it was better to implement a less ideal solution that we could get to work.

5.2.1 Work on Keras and TensorFlow

As none of us had any experience working with CNN we found that using Keras would be the best way for us to do it. We trained a few models for different things to get familiar with the use of Keras and models. We also found that Project_MYM (Underwood, 2020) and Bakkenbaeck (Saurabh, 2019) had used CNN through Keras. We tested out using the same models as them, but we could not get a model that would work correctly. While training the model it had an accuracy of about 97%, however when we applied it to an image it categorized all the images to the same category.

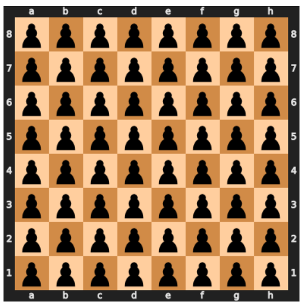


Figure 5.1: This is an example of a prediction using CNN on a board in starting position

We tried using different base model, freezing and unfreezing layers and changing the epoch steps and validation steps. None of this worked, so in the end we ended up having to give up this solution as we got pressed for time. We tested with the base models ResNetV2, VGG16 and VGG19 and got the same problem from all of them. The reason we chose these was that we found the similar projects had chosen these.

Keras datasets

For the Keras datasets we looked at many other projects and tried to find a usable dataset for our model. We eventually found some datasets that we used to train our model. We. The main dataset we used was a very diverse set containing images of many different types of chess boards and pieces. We got this dataset from from Saurabh's project from Bakken & Bæck. We would like to make our own dataset using the camera and setup at the chess club so that we could have it trained heavily in the surroundings that it is meant for, however as the camera was not available in time this was not possible.

5.2.2 Detecting piece contours

After abandoning the CNN we found that locating the position of each piece on the board would be enough to keep track of the pieces. As we know the starting positions and where all the pieces are we can match the pieces as described in chapter 4.5. This proved to be a viable solution, however it is also quite fragile. It is based on reading the size of the largest contour in the image, and is therefor a lot more sensitive to disruptions, such as glare from light, or a hand. With all variables set specifically for the surroundings and and the conditions meet it will work just as well as a CNN would, and be a bit faster.

5.3 Move detection

To detect a move we locate the square that was previously occupied, but after the move is free. From that we match the board to fit a reformatted FEN as described in chapter 4.5. This method is very dependent on the piece detection to be accurate as it is solely based on the position of the pieces. If there is a mistake from the piece detection then this will no longer function. This also has a weakness when a pawn is transformed into another piece, as we can find the pawn has reached the other side, but not what it has transformed into.

5.4 Implementation of the project

Due to a lack of a front-end in this project, the code needs to be run through Python. This can cause some issues if the computer running the code does not have all of the dependencies installed. In order to successfully run the code, Python will need to have these installed:

- OpenCV

- Python-chess
- svglib
- Chessnut

All of these dependencies can be installed directly in Python by opening the Python command prompt and entering the commands for installation.
OpenCV can be installed by entering: ***pip install opencv-python***.
Python-chess can be installed by entering: ***pip install chess***
svglib can be installed by entering: ***pip install svglib***
Chessnut can be installed by entering: ***pip install chessnut***

When all these dependencies have been installed, and assuming that the required Python files are present, the project can run.

5.5 The use of Jira

Jira has been used by the group since the beginning of the project, and has helped us work towards planed goals.

5.5.1 Sprints

The sprints in the project have been worked on in a length of two weeks, where issues were created at the beginning and were planned to be completed by the end of each sprint. Issues in the beginning of the project were created and completed in accordance to our pre-project plans time estimate, as the early parts of the project was easy to determine and predict the amount of work and length needed to be completed.

5.5.2 Backlogs

When a planned issue or task was not completed during the current sprint it was moved to the backlog to be worked on during the next sprint. Later on in the project the work on the planned issues and tasks from our pre-project plans time estimates became more and more held back as obstacles in the form if inexperience and lack of knowledge with the planned issues occurred. Some issues and tasks were dependant on the completion of earlier issues and were not stared on until these earlier issues had been resolved which resulted in several issues started after their anticipated start date.

5.5.3 Assignments

During the project issues and tasks were assigned to the group member which was best suited to work on them. This meant that issues and tasks revolving around a certain sections of the code would be handled and worked on by the group member with the best knowledge of that section. If a group member was stuck on a issue, researching potential fixes and methods usually helped but if little to no progress was made, other group members could help with those issues.

5.6 The use of git

The use of git throughout the project proved little to no issues. One issue we faced was the fact that there was a limit to the size one can upload to a repository. When trying to upload a dataset to the repository, we were informed that the size of the upload was to large, and could not be completed. This was of little issue as we managed to use a cloud for uploading and sharing the dataset, google-drive.

5.7 Communication

This section will describe the communication between the different parties within the project. The organization of the project was discussed more in depth in chapter 3.1. This section will focus more on how the communication between the different parties played out.

5.7.1 Group members

The communication between the project group members has been primarily done via digital means. Due to the current situation revolving around the national and regional preventive measures for the COVID-19 situation, the group has primarily been working from their own homes. As mentioned in chapter 3.1, most of the members of the group have been working part time jobs with dynamic working times, which made a pre-determined working period harder to follow. Ideally, we would have daily meetings, but as mentioned it did not work out for us. We were at least able to have one meeting per week, to write a status report and discuss the progress on the project.

5.7.2 Client

The communication between the project group and the client has proceeded in the form of pre-determined meetings and e-mails. We had three main meetings with the client and the rest of the communication happened through e-mails.

We had the first meeting with the client very early on, while we were working on the pre-project plan. It was during this meeting where we really got a full picture of what the client wanted the end product to be like. We discussed some more minor topics like the fact that the program should output the PGN at the end, and we also got some advice. However, the most important topic for this meeting was that we were informed that the camera should preferably be mounted to the roof. When we started thinking initially after receiving the assignment, we were planning on having the camera at the side of the chessboard, slightly elevated. This way we would be able to get good footage of each piece, making it possible to use image analysis and train the program to recognize each piece. The client, however, wanted the camera to be mounted to the roof, so that it would not be in the way of the players. We quickly realized that this would be much more tricky, since most chess pieces look very similar from above. This meant that we had to rethink out some of our initial ideas, which means that this meeting the most influential.

The second meeting was not anything spectacular, we had this meeting around the midway point of the project. We told the client about which methods we had decided to use, and the progress we had made in order to to keep him up to date. In return the client advised us to not perfect the program too much the entire way, and instead get each step working fine, and then rather perfect it at the end. We also discussed visiting the chess club to test our program there if the COVID-19 restrictions were lifted. In the end, the restrictions were never lowered to the point where we found it necessary to visit the chess club. Instead the client visited the club for us to take some pictures for us.

It was when we came to the third meeting that we had discovered that the solution using CNNs, was not working optimally, so we informed him about the alternate solutions we were working on. We also told the client that we only got the program to work with one chessboard. This problem is further expanded upon in the previous chapters. The client informed us that he had ordered a camera and a computer, but that they were going to arrive after the deadline of this thesis. We also discussed several other minor topics.

5.7.3 Adviser

The communication with the groups adviser have been going according to plan. A meeting has been held after every sprint with the projects adviser, and every group member has been present in the majority of them. Sometimes the meetings have been rescheduled, but only to a later time that day, and never to a different day. The main topics for the meetings were always what work we had done during the last sprint, what challenges we had encountered and what we were going to do for the next sprint. The meetings were always useful to us, since whenever we felt that we needed advice, we were able to get it.

5.8 Time estimates

In the pre-project plan we made a rough estimation of how we thought we might use the time over the course of the project. Over the course of working on the project there has been changes made to this plan, since some tasks took longer time than expected. Even so, the initial estimation we made helped us throughout the project to have a rough idea of how long we should spend on each task.

5.9 Changes in equipment

In chapter 3.11 we mentioned the equipment we were planning on using for the project. For different reasons we had to make some changes during the project.

5.9.1 Camera

The camera we were going to use at the chess club is described in more detail in section 3.11.1. The main reason that we were not able to use this camera, was because of the COVID-19 situation. Since we were not able to visit the chess club, we were also not able to set up our program there with the planned camera. Additionally, the client waited with ordering the camera until towards the end of the project, which meant that the camera did not arrive until after the deadline of the project anyway.

For the images we ended up taking ourselves we used a LG V30 phone's camera with the resolution set to 1080p and landscape orientation.

5.9.2 Computer

Similarly to the camera, we were not able to use the computer at the chess club. The computer we were going to use at the chess club is mentioned in more detail in section 3.11.2. Instead of using the computer at the chess club we used our personal computers to run the program.

5.9.3 Chess set

In section 3.11.3 we mentioned that we planned on using the chess set at the chess club. We did end up using the chess club at the club, since the client took pictures of this chess set for us. We also ended up using other sets of chess that we had personally, in order to get different pictures according to our needs during the development of the project.

Chapter 6

Conclusion

In concluding the work on this project, we as a group have gained newfound knowledge and experiences in a variety of areas. The group chose this project due to the fact that each member had previous experiences with image analysis. The project has provided the group with challenges and newfound experiences within the use of advanced image analytics and convolutional neural networks that extends farther of any previous knowledge and experience we had practiced before the project.

We have tried approaches involving creating solutions to the projects issues from scratch and learned that it can be advantageous to take inspiration from and or base our work on the work done by others similar projects. From this we have managed to create a system that is capable of detecting a chessboard from an image or more precisely a frame from a video of reasonable dimensions, and individually detect and process each tile along with any chess piece that may stand on it. This has been achieved by training CNN models for predicting what chess piece stands on which tiles. The finished system will be able to be operated from a laptop at the chess club with a connection to the web-camera mounted on the ceiling.

Due to the changing COVID-19 situation, testing of the system in action at the chess club have not been possible. These testings would revolve around the positioning of equipment and potential optimization of some sections of the code based on said positioning.

Chapter 7

Bibliography

Electronic Boards (2021) Available at: `http://www.digitalgametechnology.com/index.php/products/electronic-boards?mavikthumbnails_display_ratio=2` (Accessed 05/04/2021).

Om klubben (2021) Available at: `http://alesundsjakk.no/klubben/` (Accessed 06/04/2021).

Saurabh, B. (2019) *Convert a physical chessboard into a digital one*. Available at: `https://tech.bakkenbaeck.com/post/chessvision` (Accessed 09/04/2021).

Convolutional neural network (2021) Available at: `https://en.wikipedia.org/wiki/Convolutional_neural_network` (Accessed 16/05/2021).

Sjakk i Ålesund (2021) Available at: `https://alesundsjakk.no/` (Accessed 09/05/2021).

Manifesto for Agile Software Development (2001) Available at: `https://www.agilealliance.org/agile101/the-agile-manifesto/` (Accessed 19/05/2021).

Understanding Agile Scrum in 10 minutes (2021) Available at: `https://www.tuleap.org/agile/agile-scrum-in-10-minutes/` (Accessed 19/05/2021).

Why TensorFlow (2021) Available at: `https://www.tensorflow.org/about/case-studies` (Accessed 03/05/2021).

Introduction to Tensors (2021) Available at: `https://www.tensorflow.org/guide/tensor` (Accessed 03/05/2021).

About Keras (2021) Available at: `https://keras.io/about/` (Accessed 03/05/2021).

About (2021) Available at: `https://opencv.org/about/` (Accessed 03/05/2021).

python-chess: a chess library for Python (2021) Available at: `https://python-chess.readthedocs.io/en/latest/` (Accessed 04/05/2021).

Git (2021) Available at: `https://git-scm.com/` (Accessed 05/05/2021).

Sourcetree (2021) Available at: `https://www.sourcetreeapp.com/` (Accessed 05/05/2021).

Confluence (2021) Available at: `https://www.atlassian.com/software/confluence` (Accessed 17/05/2021).

Gliffy (2021) Available at: `https://www.gliffy.com/` (Accessed 17/05/2021).

Overleaf (2021) Available at: `https://www.overleaf.com/about` (Accessed 17/05/2021).

LaTeX: a document preparation system (2021) Available at: `https://archive.org/details/latex00lesl` (Accessed 17/05/2021).

J.D (2019) How can I select ROI in a separate image in opencv-python? Available at: `https://stackoverflow.com/questions/55188572/how-can-i-select-roi-in-a-separate-im`
(Accessed 18/02/2021).

Underwood, A (2020) Board Game Image Recognition using Neural Networks
Available at: `https://towardsdatascience.com/board-game-image-recognition-using-neural-netwo`
(Accessed 24/03/2021).