MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Chess Position Recognition from a Photo

Master Thesis

Zoltán Orémuš

Brno, spring 2018

Declaration

I hereby declare that this master thesis is my own work and that the information I used has been fully acknowledged in the text and included in the reference list. I agree with putting the thesis on public display at Masaryk University for study purposes.

Zoltán Orémuš

Supervisor: doc. RNDr. Pavel Matula, Ph.D.

Acknowledgement

Abstract

Object recognition is a useful subject in the field of image processing and computer vision. To achieve this goal, one must find the object in the image and classify it, often by the means of similarity to other known objects. If the object's location in the scene is known, its location in the image can be calculated using parameters of the camera, deduced from 3D to 2D point correspondences. In this work, the objects needed to be classified are chess pieces standing on a chessboard. Purpose of this thesis is to utilize chessboard detection, camera calibration and object classification to calculate chess positions from a single image or a group of images. Existing image processing algorithms are put to use to create a novel approach where chess pieces at each position of the chessboard are classified based on their similarity to contours generated from 3D models of the pieces at the respective positions.

Keywords

Table of Contents

1 Introduction

Chess is a game played by millions of people all over the world with a long tradition. It is widely used as a pastime, an afterschool activity or even as a competitive sport. There are some former world chess champions that many people, even those who do not play chess, know about. As it is so popular, there have been many instances of trying to automatize it. Many years of research went to creating supercomputers able to beat people. In the beginning, it was just a fantasy but as computers became faster and algorithms more sophisticated even the strongest chess players started losing against them [1]. Not even the greatest grandmaster can calculate and prune options for as many moves as a computer can. Chess applications became common on home computers and became a standard package with some operating systems. Online chess communities are rapidly growing and websites such as chess.com host more than one million chess games every day [2]. The demand for chess is large enough for development of various programs that would serve as a helpful guide for the betterment of a player. Engines exist which can analyze the best moves and say where the player has made a mistake. Analysis of a position thus becomes a vital part of the learning curve.

Imagine a player who is at a tournament and gets into an interesting position that would require later analysis. Various computer engines can do it, and they offer the player many ideas as to how the situation could have been handled. The engine has to be fed the position of the pieces, but first, this position has to be somehow recorded. In some tournaments everything is written down, but in fast games, there might not be the time to do so. In such cases, the player could snap a photo of the chessboard and download it into a computer at home. A program could then recognize the positions in the photograph and feed it to the engine for the analysis. This photograph should be taken with care, so all the pieces on the board are visible and the angle is not too low, so the pieces don't occlude one another, or too steep, so the shape of the pieces is as recognizable as possible.

This thesis proposes a program for recognition of chess positions from a single or several photographs and saving it in a generally known chess format.

The recognition is done without machine learning algorithms for classification of the pieces but rather by using more traditional image processing methods. The program is tailored to work with Czech Club pieces. First, the chessboard in the image is detected using Hough transformation. The chessboard has 64 squares and each of them can be occupied by a piece. Image of each of the piece on the chessboard is deformed by a projection matrix given by camera parameters. These parameters are calculated from correspondences of points from the detected chessboard in the image space to the points of the chessboard in the world space. This calculation leaves us with known position and rotation of the camera, as well as its inner settings, such as focal length. Instead of having a massive dataset of all the pieces in various positions, all of the six pieces are modeled and then rendered on each of the 64 positions of the chessboard from the previously calculated viewpoint of the camera. This creates the templates of the six pieces for each of the 64 positions of the chessboard, all of them projected in the same way as the pieces in the photograph. The chessboard is then analyzed by calculating the color of each square, its occupancy and the color of the pieces. If a square is occupied it is then compared to the six templates that were rendered at the respective position in order to classify the type of the piece.

The second chapter of the thesis deals with an overview of the methods used in this thesis, the third gives some outlines of the chess rules that posed constraints on the thesis and the fourth chapter deals with the creation of the dataset. The algorithm itself is described in the fifth chapter, split into parts about chessboard recognition, camera calibration, generation of the templates and analysis of the positions and the pieces. At last, there is the evaluation of the algorithm's performance and conclusion dealing with possible further expansion of the work.

2 Overview of current methods

There are three major tasks to deal with when one wants to recognize chess positions from a photograph. The first task is to recognize a chessboard in the image and analyze its orientation. The second task is a calculation of the location of the camera and its rotation so we know how the pieces are projected in the picture, whether it is from a top view, or a side view, or any other view, which can be very helpful during the classification process. The third task is to classify the pieces on the chessboard.

This work makes use of many known methods to achieve its goal. The problems of chessboard detection, camera calibration, and classification based on similarity to other objects have been widely researched. A large part of the thesis relies on the calculation of the parameters of the camera that took the photograph. The calculation of these parameters is based on the chessboard pattern in the photograph. Detection of the chessboard is often done using either the corners with the Harris detector [3] or the lines with the Hough transformation [4] the chessboard consists of [5]. In this thesis, Hough transformation is utilized. Polar representation of a line was chosen so the Hough transformation does not suffer from limitations of parametric equation $y = kx + q$ where in the case of a vertical line, the parameter k goes to infinity [6]. The axes of the Hough space represent the parameters of polar representation of a line ρ and θ. θ-axis goes from 0 to 180 degrees. The rest of a circle, 180 to 360 degrees is covered by negative values of ρ. Each pixel $[x, y]$ of interest from the image is represented as a curve in Hough space given by the equation

$$\rho = x \cdot cos(\theta) + y \cdot sin(\theta)$$

The coordinates of intersection $[\theta, \rho]$ of any two curves in the Hough space signify the parameters θ and ρ of a line passing through the two pixels in the image these two curves represent. When a number of pixels lay in line in the image, all of their curves in the Hough space intersect at the same point. If the

curves are calculated from pixels representing edge map of the image, the coordinates in Hough space where many curves intersect represent a long line of pixels in the edge map.
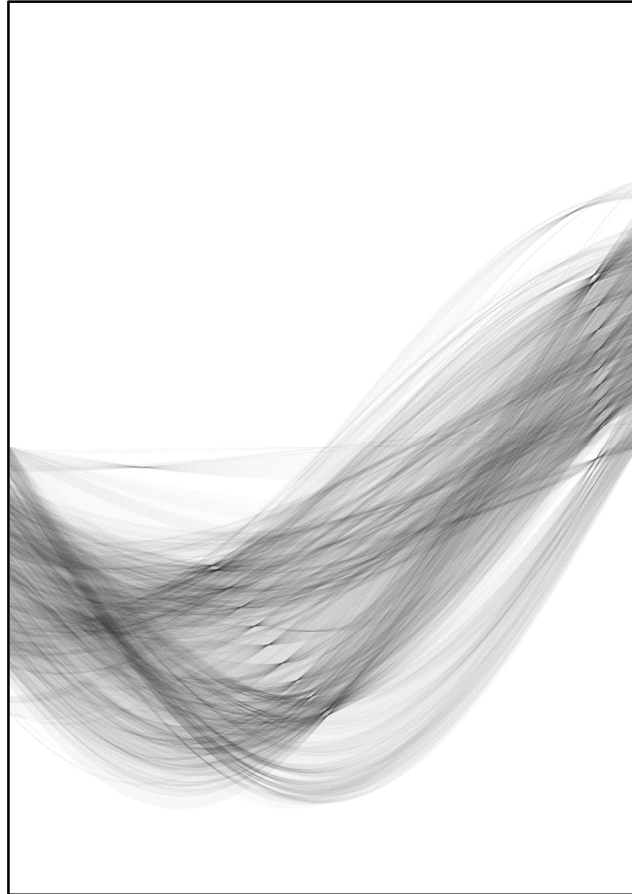

Figure 1:
An example of Hough space. Every curve represents a pixel in the image and every intersection of curves represents a line passing through pixels that represent these curves.

The input photograph is captured by a standard camera. It is a camera that a person capturing the game has currently on him, very often a cell phone camera or some sort of compact. This kind of camera makes a perspective projection of the world and creates a 2D image. If the image has enough information and the scene contains well-known objects, a lot of information can be deduced about the camera. We distinguish intrinsic parameters and extrinsic parameters of the camera. Intrinsic parameters are specific to a camera. They

include information such as focal length (f_x,f_y) and optical centers (c_x,c_y). They rely on the camera only, so once calculated, they can be stored for future purposes. Extrinsic parameters correspond to rotation matrix and translation vector which specify the position of the camera in the scene [7].

Both intrinsic and extrinsic parameters are contained in a camera matrix M. This matrix is calculated from a set of image-point correspondences, i.e., from a set $\{(u_i, X_i)\}_{i=1}^{m}$, where u_i are homogenous 3D vectors representing image points and X_i are homogenous 4D vectors representing scene points. Homogenous linear system

$$\alpha_i \boldsymbol{u}_i' = M\boldsymbol{X_i}, i = 1, \dots, m$$

needs to be solved for M. M is calculated by linear estimation by minimizing the algebraic distance. Multiplying equation $\boldsymbol{u} \simeq MX$ by $S(\boldsymbol{u})$, where

$$S(\boldsymbol{u}) = S([u, v, w]^\mathsf{T})] = \begin{pmatrix} 0 & -w & v \\ w & 0 & -u \\ -v & u & 0 \end{pmatrix}$$

from the left makes left-hand side vanish, yielding $0 = S(\boldsymbol{u})MX$. Rearranging this equation yields

$$[X^\mathsf{T} \otimes S(\boldsymbol{u})]\boldsymbol{m} = \boldsymbol{0}$$

where $m = [m_{11}, m_{21}, \dots, m_{34}]^\mathsf{T}$ and $\otimes$ is a Kronecker product. Considering all m correspondences yields the system

$$\begin{bmatrix} X_1^\mathsf{T} \otimes S(\boldsymbol{u_1}) \\ \dots \\ X_m^\mathsf{T} \otimes S(\boldsymbol{u_m}) \end{bmatrix} \boldsymbol{m} = W\boldsymbol{m} = \boldsymbol{0}$$

We minimize the algebraic distance $\|W\boldsymbol{m}\|$ subject to $\|\boldsymbol{m}\| = 1$ by SVD. As a next step, M is decomposed to intrinsic and extrinsic parameters. Matrix K represents intrinsic parameters. It is expressed in the form of 3 by 3 camera matrix [7] [8]

$$\begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix}$$

Matrix $[R| - Rt]$ represents extrinsic parameters where R is a 3×3 matrix describing rotation and $-R\boldsymbol{t}$ is a 1×3 vector describing translation. Projection matrix M is a 4×3 matrix, containing all intrinsic an extrinsic parameters in the form of

$$M = K[R| - Rt]$$

The decomposition of M to $K, R, and\ t$ is unique. Denoting $M = [A|\boldsymbol{b}]$, we have $A = KR$ and $\boldsymbol{b} = -A\boldsymbol{t}$ where $\boldsymbol{t} = -A^{-1}\boldsymbol{b}$. Decomposing $A = KR$ where K is upper triangular and R is rotation can be done by RQ-Decomposition, which is similar to QR-Decomposition [9].

This general calculation is simplified by the fact, that the intrinsic parameters can be calculated just by parsing the data about the camera information from the image. Only the extrinsic parameters are the ones that need to be calculated from the correspondence between the scene and the picture. Obtaining the intrinsic parameters is done by parsing the EXIF data from the picture and then by calculating the parameters from this data. All of the necessary information is stored for later use. The information includes image height h and width w and focal length f. Another necessary information to calculate the camera parameters is the sensor size. This information is missing in the test dataset so it has to be put in manually.

Focal length (fx, fy) and optical centers (cx, cy) for the matrix K need to be calculated. cx and cy are centers of the image

$$cx = w/2$$
$$cy = h/2$$

In order to have a correct camera matrix, all of the parameters need to use the same units. The focal length provided by the EXIF information is in

millimeters, while cx and cy are in pixels. The following equations are used to get them into the same unit

$$fx\ (pix)\ =\ f(mm)\ \cdot\ w(pix)\ /\ Sw(mm)$$
$$fy\ (pix)\ =\ f(mm)\ \cdot\ h(pix)\ /\ Sh(mm)$$

where Sw represents sensor's width in millimeters and Sh represents sensor's height in millimeters. Consequently, fx is often equal to fy, because the ratio of w to h is equal to the ratio of Sw to Sh for many cameras. OpenCV function used to calculate this entire process also calls for a vector of distortion coefficients. For the purpose of this thesis, it was not necessary to include these coefficients, as the images taken by widely used cameras, such as cell phone cameras, do not get distorted much. For this reason, the vector of distortion coefficients is filled with zeros. The result is still highly accurate, as will be demonstrated in later chapters. The extrinsic parameters holding the information about the location and rotation of the camera can now be determined from intrinsic parameters and corresponding points from the world space to the image space by solving and decomposing the camera matrix M.

The problem of piece recognition has already been approached several times. The difference is that most of the other works dealing with this issues do not generate the dataset for classification but rather rely on creating a dataset by photographing the pieces from many angles and then labeling them. Papers such as [10] and [11] deal with piece recognition and approach it in such manner. They get quite solid results but since there is no general database of pictures of chess pieces, an approach with generating a classification dataset from 3D models was chosen for this thesis. Creating 3D models of six pieces is not a difficult task for a skillful modeler and in this case, it took about 30 minutes, which is much less than photographing pieces from various angles, extracting them out from an image and labeling them. The upside of such approach, in contrast to training convolutional networks [12], or similar machine learning structures, is that a large dataset of images does not have to be stored and nothing needs to be trained. The pieces on the chessboard only need to be compared to generated images to determine their class.

Several variants of classification were tried, all of them relying on template matching. Template matching is a technique for finding areas of an image that match (are similar) to a template image. Template matching in this thesis is done in two variants. One variant applies normalized cross-correlation, where a template T is laid over every possible position in the image I and a response at the position $R(x, y)$ is calculated using the function

$$R(x, y) = \frac{\sum_{x',y'}(T(x',y') \times I(x + x', y + y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \times \sum_{x',y'} I(x + x', y + y')^2}}$$

The location with the best response from the cross-correlation is considered to be the best match signifying that the area hiding beneath that location best resembles the template. The other way of calculating the location of the best match is by using the phase correlation. Phase correlation finds relative offset between two images [13]. If we have a template we need to search for in an image, the phase correlation can be used to find it. It expects two images g_a, g_b of the same size, so the template is fitted into the center of a black image with the same size as the image in which we are looking for the best match. First, a window function (e.g., a Hamming window) on both images is applied to reduce edge effects (this may be optional depending on the image characteristics). Then, discrete 2D Fourier transform of both images is calculated.

$$G_a = \mathcal{F}\{g_a\}, G_b = \mathcal{F}\{g_b\}$$

The cross-power spectrum is calculated by taking the complex conjugate of the second result, multiplying the Fourier transforms together elementwise, and normalizing this product elementwise.

$$R = \frac{G_a \circ G_b^*}{|G_a \circ G_b^*|}$$

Where   is the Hadamard product (entry-wise product) and the absolute values are taken entry-wise as well. The normalized cross-correlation is obtained by applying the inverse Fourier transform.

The location of the peak in   is determined.

now signify the offset between the two images [14]. Following Figure represents an example of two shifted images and image   where the location of the peak represents the offset between the images.
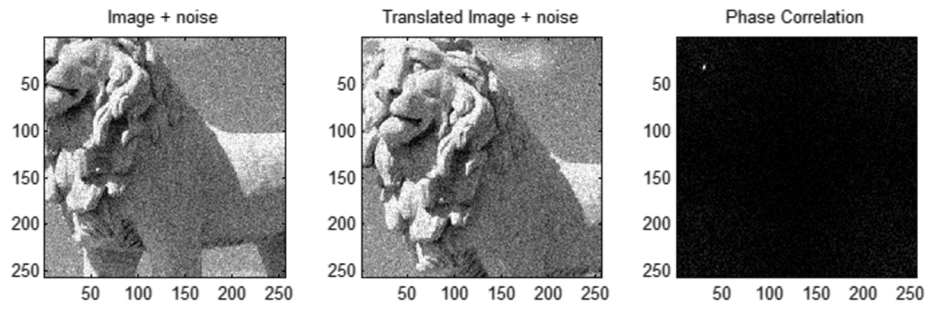

Figure 2: Two images and their phase correlation signifying their relative offset.

3 Constraints given by chess rules

The chess rules pose some constraints on piece positioning, the board orientation, and give some requirements for the form of the output.

There are never more than eight pawns of each color and there is precisely one king of each color. In case the classification returns more pieces than there are, the algorithm must filter the findings, where every non-logical piece is cross-referenced with the rest of the pieces. For example, given the similar contours of the rook and the pawn, if the algorithm returned more than eight pawns of one color it is possible that one of these pawns was a rook, wrongly classified as a pawn.

Pawns can never stand on the first or the eight rank. When a player moves a pawn to the rank at the end of the board, he must exchange that pawn as part of the same move for a new queen or a rook or a bishop or a knight of the same color on the intended square of arrival [15].

When a piece is promoted, the most common choice is to promote it into a queen. The queen moves as both the bishop and the rook combined. For this reason, it seems unlikely that there would be more than two rooks and two bishops of each color on the chessboard, but there might come up a situation when promoting to queen results in a stalemate and the player would rather choose either a rook or a bishop. Figure 3 shows a sample position of such situation.
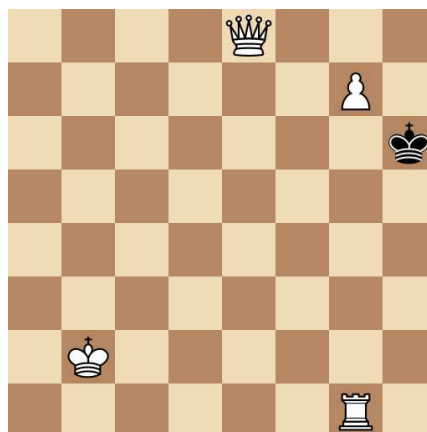

Figure 3: White's move, if white moves the pawn and promotes it to a queen, stalemate ensues

The movement of knights might offer a tactical advantage and such promotion is quite common. The only constraint supplied by the promotion is the no-pawn-in-the-back-rank constraint described in the previous paragraph.

The chessboard is always rotated so the rank closest to the player has a white square on the right hand-side. This is a useful feature that helps to decide which way the chessboard is oriented.

4 Dataset

The dataset for the thesis consists of either individual photographs of a chess game or sets of photographs of the same position from various angles. The nature of this work requires certain precautions to be taken and constraints to be posed when capturing the images. The main problems include occlusions, lighting issues, quality of the photograph and the angle and the position from which the photograph is taken.

4.1 Chessboard and pieces

In order to try out the viability of this algorithm, it was decided to use standardized chess equipment to limit classification problems. The chessboard is $6cm \times 6cm$ square-sized board and the pieces are Czech club pieces [16] with the following heights

Piece	Height (cm)
King	10.5
Queen	10
Rook	6
Bishop	8
Knight	7
Pawn	6

Table 1: Heights of pieces in centimeters

The reason for using this chess set is its common use in the Czech Republic and its availability. A large number of official games is played with this configuration chessboard and these exact pieces. The algorithm does not currently work with other chess sets as it uses comparisons to these pieces. They are shown in Figure 4. All of these pieces were also modeled by hand in 3D software Blender.

Figure 4: Pieces, from left to right, top to bottom: pawn, bishop, knight, rook, queen, king

4.2 Images

As in other computer vision problems, a hidden information about occluded objects is a problem. In other applications, there is a possibility of guessing what the occluded object might be or even guessing the existence of a fully occluded object if there is some prior knowledge of the scene. It is not so in chess. In wrong angles, small piece standing in a square adjacent to one occupied by a large piece might be partially or even fully occluded by the said larger piece. In such cases, wrong recognition is expected. It is impossible to guess chess piece positions because, except for a few notable exceptions (the pawns), any piece can occupy any square.

Piece-to-square occlusion is also often present. If a piece is positioned very close to another square which is empty, a majority of the piece is projected over that square. In those instances, a large portion of the empty square is occluded by the piece and the algorithm might classify it as occupied.

The best practice is to choose the angle from which the photo is taken appropriately, so such issues have as little occurrence as possible. The photographer should take the picture in such a way that entire chessboard is in the image, ideally in its center and have most of the pieces as visible as possible.

A large prerequisite in any image recognition are well-lit scenes. Black areas and white areas create homogenous blobs where any recognition is

impossible. This is especially notable in the case of dark pieces standing on dark squares. If the scene is too dark, the dark piece and the dark square appear almost black and it is impossible to recognize them from one another. On the other hand, sharp light hitting the board ends up creating undesirable patterns as it hits the board's imperfections resulting in other problems, such as undesirable edges.

The quality of photographs for this program is not too demanding. All that is necessary is a steady hand and well-focused lens which results in a sharp photograph. Blurred photos are not desirable as they spoil edge detection which is a vital part of the algorithm.

Pictures taken by cameras contain EXIF. It is named for the DCF standard Exchangeable Image File Format (EXIF), which stores interch3ange information in image files [17]. EXIF information is necessary for computation of the projection matrix.

Figure 5: An example of an input image

5 Algorithm

As mentioned in the introduction, the algorithm itself consists of several building blocks that depend on one another. The first step is chessboard recognition. It also serves as a sieve, which stops the program when a chessboard of 64 squares is not recognized. Hough transformation produces information about the lines of the chessboard. This vital information is later used for segmentation.

The second step is the calculation of camera's intrinsic and extrinsic parameters. Information about actual sizes of squares on the chessboard in world space combined with recognition of these squares in the picture space allows for calculation of these parameters.

The third step is feeding the camera data to 3D modeling software Blender and rendering every modeled piece at each of the 64 positions from the viewpoint of the camera. The output consists of $5 \times 8 \times 8 + 1 \times 8 \times 8 \times 8 = 832$ renders. There are 5 rotationally invariant pieces, all of them on 8×8 positions. Then there is the knight, the only piece where the rotation is important. It also needs to be rendered on 8×8 positions and in 8 different rotations. These renders are used for comparisons with the pieces in the photograph in order to classify the pieces.

The fourth step processes these renders so that only the outline of each piece remains. It serves as a template during the classification stage. It is also necessary to get the outlines of only the top half and only the bottom half of each piece in separate outputs. These outputs are utilized in different techniques.

The fifth step is chessboard analysis. Histograms are used to determine which squares are white and which are black and whether a dark or a light piece sits on a square. Combination of histogram and template matching approaches was tried to determine occupancy of a square.

The sixth step is the classification. This implementation focuses on template matching approach. An edge detector is applied to the photograph and the edges of each piece are compared with contours of the templates.

The last step is filtering improbable results and printing the final output.

5.1 Chessboard detection

A chessboard is   squares, meaning there are 9 lines going in one direction and another 9 lines going in a perpendicular direction. These 18 chessboard lines outline all of the 64 squares. In order to detect the lines, the input needs to be processed with an edge detector. Since the chessboard consists of squares with extremely different intensities, there is a large gradient on the borders between the squares. Canny detector was used to detect these edges as illustrated in Figure 6
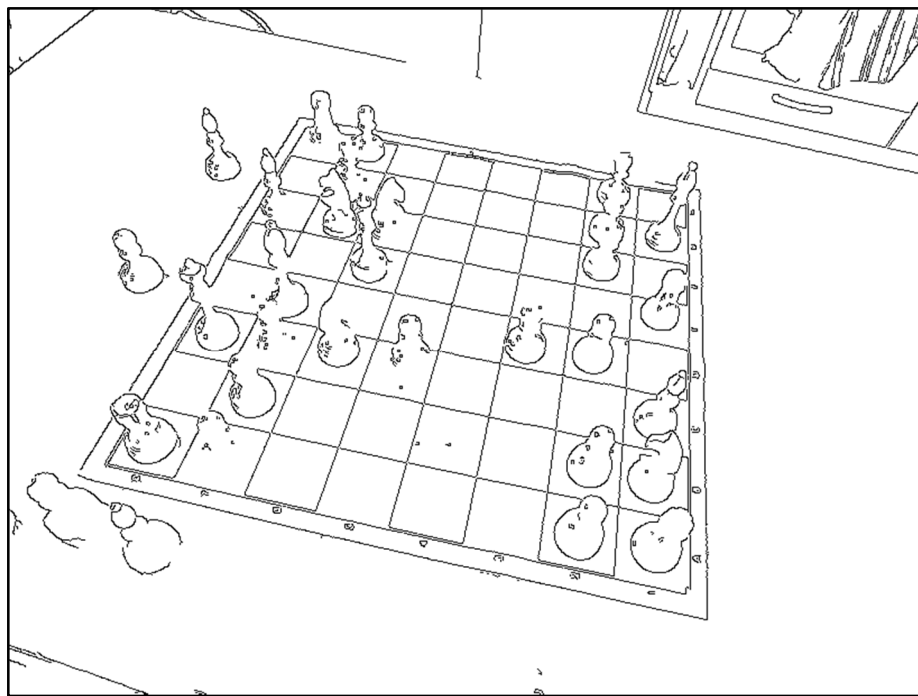

Figure 6: Input image with the Canny edge detector applied. The image is inverted for better visualization.

Undesirable edges are also detected, for example, an edge of the table the game is being played on. These edges need to be ignored as we are only interested in the 18 chessboard lines.

The Canny edge detection outputs an edge map, a binary image, where white pixels indicate the detected edges. Hough transformation is used to detect the 18 chessboard lines. An accumulator is created to represent the Hough

space. It is a matrix of zeros with the x-axis representing   and the y-axis representing . Each pixel   from the edge map is then transformed into a curve via a cycle, where we go through the entire range of   and calculate   for each of them with the equation

Each   along with calculated   gives a coordinate   in the accumulator. The value at this coordinate is incremented by one. The fact that the curves of all of the pixels lying on the same line in the edge map share the same intersection coordinate in the Hough space results in high values accumulated at the coordinates that represent the line which passes through these pixels in the edge map. These are the interesting intersections because the grid of a chessboard is made of long lines. The accumulator is shown in Figure 7.
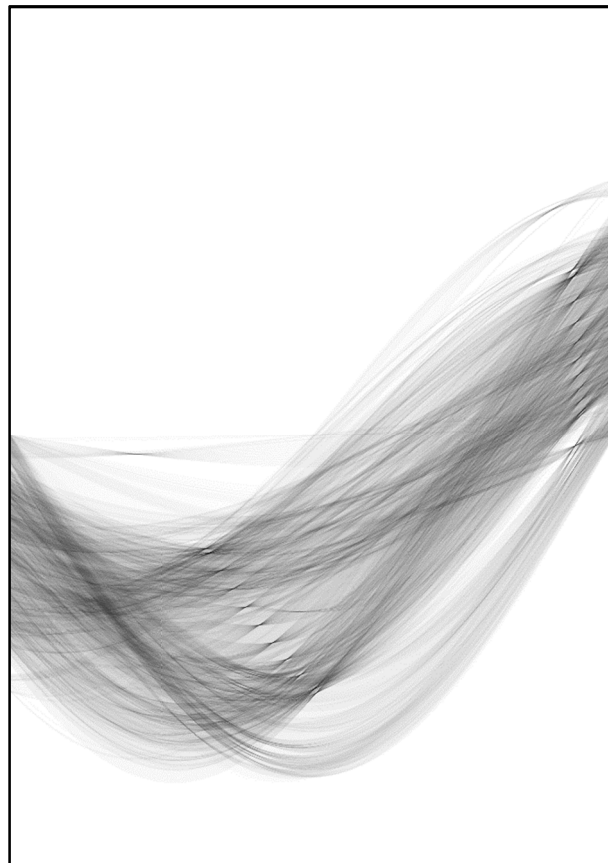

Figure 7: The accumulator representing the Hough space after the Hough transformation. The image is inverted for better visualization.

Low values in the accumulator must be removed so only the coordinates signifying long lines in the edge map remain. This is done using mathematical morphology. H-extrema transformation is used. It provides us with a tool to filter image extrema using a contrast criterion. More precisely, the h-maxima transformation suppresses all maxima whose depth is lower or equal to a given threshold . This is achieved by performing the reconstruction by dilation of the input image   from

where   represents reconstruction by dilation and   represents an image, where all the intensities are lowered by value of h. h-convex transformation is defined by subtracting the h-maxima from the original image and is applied to get the peaks
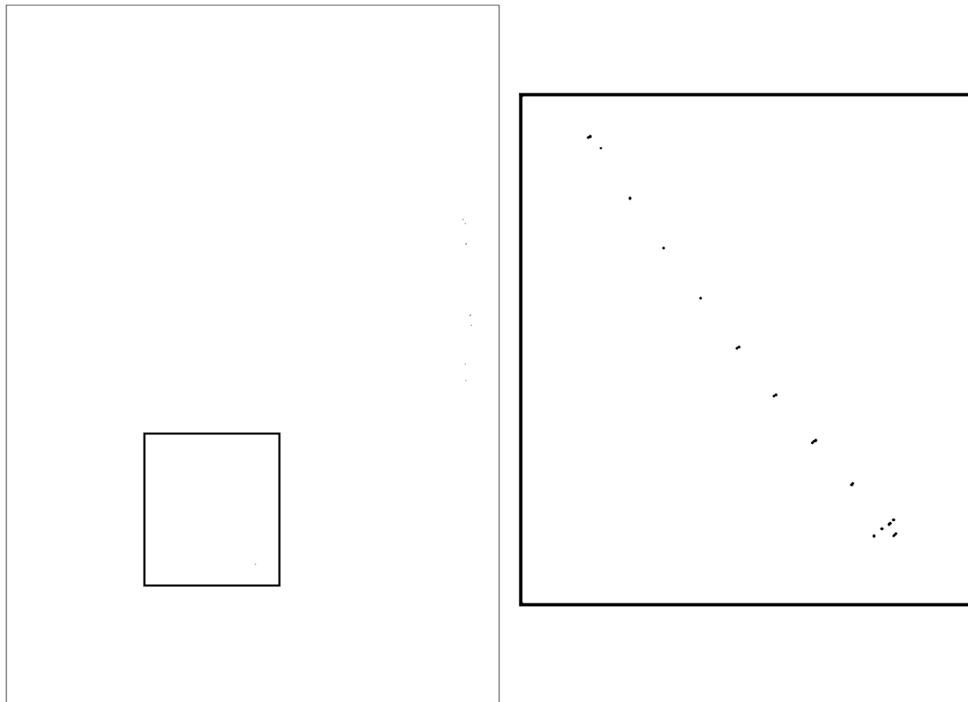
[18]

Figure 8: The accumulator after the morphological operations. The square on the right is a close-up of the region outlined by the gray rectangle in the left image. The image is inverted for better visualization.

We are looking for a single coordinate representing a line. Due to some rounding errors, the curves do not always meet in the same spot but all of them rather pass through a small area. The result after the h-convex transformation are the brightest areas based on the contrast criterion. These are the peaks representing the lines. There is a possibility that due to the rounding errors some peak area representing a line gets split into parts while removing non-significant low values and it needs to be restored. Thresholding is used, so the values higher than zero get the maximum intensity. Now the accumulator is made of white blobs. Dilation operator is applied and if a peak is indeed split apart and created more than one blob, these blobs join. Each of the final blobs now lies over the actual coordinate representing a line. The highest intensity hiding beneath each blob is found and the coordinates of this location are parameters of a polar line representation of each line in the edge map.
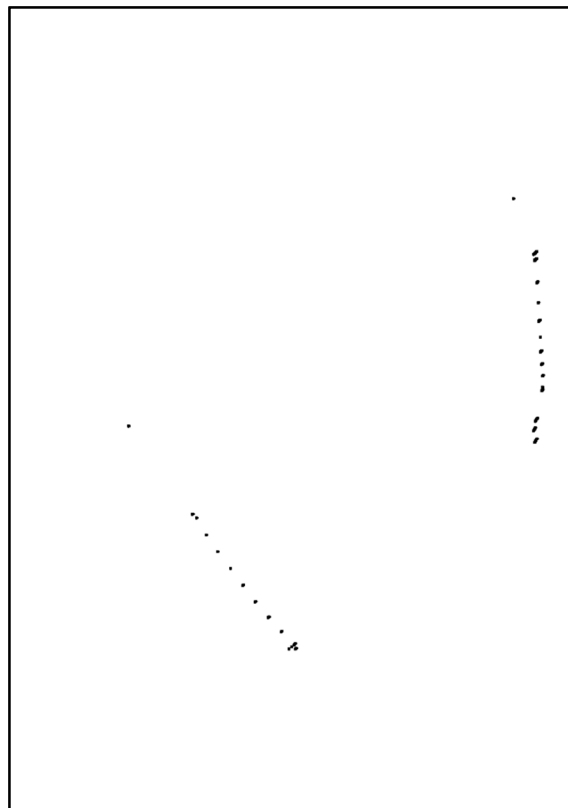

Figure 9: The accumulator is now filled with white blobs, each lying over a coordinate representing the parameters of polar line representation of each line in the edge map. The image is inverted for better visualization, so the black blobs here are actually white.

After finding the highest value in the accumulator hiding beneath each white blob, the accumulator now consists of single pixels, each representing a line in the edge map. They should represent all of the chessboard lines along with some other lines found in the edge map, such as the edge of a rectangular table. This edge is also long so it would show as a high peak in the accumulator. Now the task is to find out which 9 of these pixels represent the 9 chessboard lines that go in one direction and which 9 of these pixels represent the 9 chessboard lines going in the perpendicular direction.

If the photograph is not taken from a low angle, one group of 9 lines have their θs approximately 90 degrees higher (or lower) than the other group of 9 lines. Due to the fact that the θs in the accumulator go from 0 to 180 degrees, one group of lines is on the left side of the accumulator and the other group is on the right side. There are some cases where this is not true, for example, if the θs of one group of lines go from 85 to 95 degrees. These are special cases that can be solved.

The accumulator is split into the left and the right part at $\theta = 90$. Now all, or in special case, most, of the lines in the first group of 9 lines lie in the left accumulator and all (most) of the lines in the second group of 9 lines lie in the right accumulator. The peaks representing the group of 9 lines lie approximately on a straight line. Let's take first three lines into consideration. Their parameters are called $\rho 1$, $\rho 2$, $\rho 3$, $\theta 1$, $\theta 2$, and $\theta 3$. If the camera made orthographic projections, the ratio of $\rho 1/\rho 2$ would be equal to $\rho 2/\rho 3$ and $\theta 1$, $\theta 2$ and $\theta 3$ would be equal, so they would lie in one vertical line. Given that this is perspective projection, the θs are slightly different and also the ratio of ρs is not equal and the peaks in the accumulator that represent the group of 9 lines have increasing spacing between them. This makes peaks in the accumulator, which represent the group of 9 lines, lie approximately on the same line. Another Hough transformation is used, this time on the left and the right accumulator, to determine these 9 peaks. This creates another accumulator where coordinate with the highest value represents a line in the left (right) accumulator that passes through or at least goes by most of the peaks. Since there is a very low chance that there would be another pattern as significant as the chessboard pattern that would create many lines parallel to each other, the brightest coordinate in the new accumulator should represent the line that passes through

the peaks in the left (right) accumulator that represent the group of 9 lines. This is demonstrated in Figure 10 where a red line passes through points in the left and right accumulator.
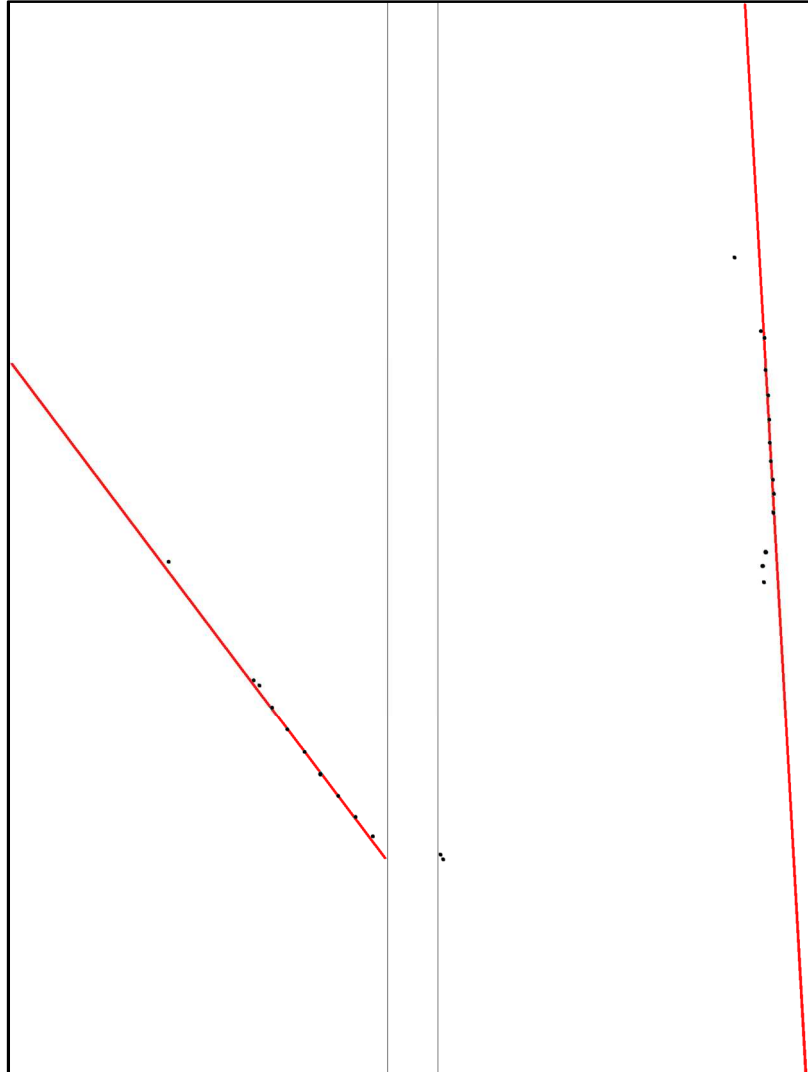

Figure 10: The left image represents the left half of the accumulator, the right image the right half of the accumulator. The pixels representing the chessboard lines lie approximately on the same line represented in red color. The image is inverted for better visualization.

As described above, a situation might occur, that a group of 9 lines has in a range that includes 90. When the accumulator is split into halves, at ,
some of the 9 peaks might fall in the accumulator covering -range of

and some might fall in the accumulator covering θ-range of $< 90, 180)$. The red lines calculated in the left (right) accumulator therefore must compared with peaks in the whole accumulator. The same situation arises when the range of θs for a group of 9 lines goes over 180 degrees and starts again at 0. Peaks in the accumulator falling within a certain distance of the red line calculated in the left accumulator belong to one group of 9 lines, while the pixels falling within a certain distance of the red line calculated in the right accumulator belong to the other group of 9 lines.

Outlier filtering is quite straightforward. The peaks in the original accumulator that do not lie within a certain distance to the red lines from the second Hough transform are not considered a part of the peaks representing the chessboard lines and are deleted.

In the case there is a peak that lies within the allowed distance of the red line and is not a part of the chessboard, for example, an edge of a table aligned with the chessboard, it is filtered out by the analysis of the distance between neighboring peaks. As mentioned above, the ratio of distance is not the same, and the distance between two neighbors is a bit larger (or smaller) than the distance between the previous two neighbors. To filter the outliers, a central peak from the group of peaks lying on the red line is chosen. There is a good chance it will be fourth, or fifth line in the chessboard, as not too many other lines with the same direction as the chessboard lines are present in the image. The algorithm looks in one direction from this middle peak and saves distance from the neighbor. Then it takes said neighbor and measures the distance from it to the next neighbor. If the ratio of these 2 distances falls within a threshold, these peaks are considered as part of the chessboard and the algorithm moves onto next peak. The algorithm stops when either the list of peaks is exhausted or a ratio is found that does not fit the threshold. The same procedure is then applied in the other direction from the middle and then again for the group of peaks lying on the other red line.

In the end, we are left with 2 groups of 9 lines representing the chessboard. These lines outline each of the squares in the chessboard. Figure 7 represents the chessboard with detected lines and Figure 8 represents the chessboard with highlighted centers of the squares.
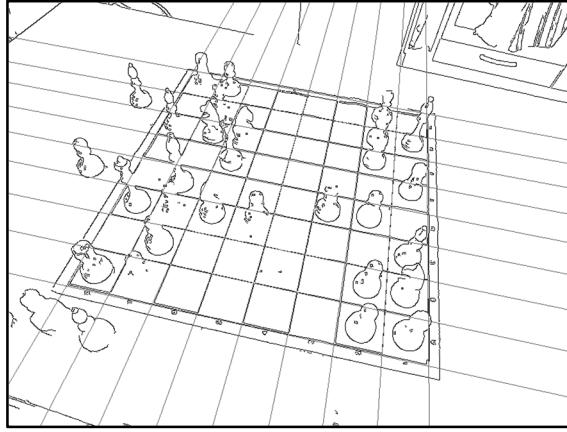
Figure 11: All of the lines representing the chessboard are detected. The image is inverted for better visualization.

Figure 12: All of the squares of the chessboard are detected. The image is inverted for better visualization.

5.2 Camera calibration

Once the chessboard is detected, the next task is to create a contour template of each piece at each position of the chessboard in the same perspective as the photograph is in. These contours are later used in template matching part of the algorithm to classify each piece standing on the chessboard. The templates are created by rendering each piece at every position in Blender. It is necessary to calculate the intrinsic and extrinsic parameters of the camera that took the photo. These are used to set up the camera in Blender and also for cutting out a region of the chessboard for each square during classification. This is done using the input image information for determining the intrinsic parameters and then

using these parameters along with the world-space to image-space correspondences to calculate the extrinsic parameters.

The decision had to be made, what points and how many of them would be used. In the end, 4 points proved enough for a very good estimation of the camera location. The 4 chosen points are the centers of the corner squares of the chessboard. Their location in the picture space is determined thanks to chessboard detection. World space coordinates are fabricated. Even though the size of the chessboard in millimeters is known, it is not necessary to use this information, all that is obligatory is to choose some unit scale and stick with it. In this case, Blender units were used, with the side of the chessboard being 4 units long, so each square is 0.5x0.5 units long. The coordinates of the centers of the corner squares of the chessboard in the world space are thus [-1.75, 1.75, 0], [-1.75, -1.75, 0], [1.75, -1.75, 0], [1.75, 1.75, 0].

In real, life 4 units would mean 48 centimeters, so one Blender unit is 12 centimeters. The number 1.75 is used because if the center of the chessboard is at [0,0,0], meaning the centers of the corner squares are 1.75 units from the center.

In order to get the intrinsic and extrinsic parameters, the method for calculating camera matrix M described in chapter 2 is used. It is implemented in openCV function solvePnP, which finds a camera pose from 3D-2D point correspondences [9]. As an input, it requires all the previously pre-computed data, the camera matrix, the vector of distortion coefficients and corresponding vectors of 2D points and 3D points. It outputs a translation vector $tvec$ and a rotation vector $rvec$ which bring points from the model coordinate system to the camera coordinate system.

These data need to be further processed for their use in Blender. Blender takes quaternion to set the camera rotation, and a translation vector to set the camera location. To get these, $rvec$ is converted to rotation matrix using openCV function Rodrigues [19]. A quaternion q is calculated from the rotation matrix. Suppose $q = (x, y, z, w)$ where w is the real part. Then the rotation matrix R corresponding to q takes the following form

$$R = \begin{pmatrix} 1 - 2y^2 - 2z^2 & 2xy + 2zw & 2xz - 2yw \\ 2xy - 2zw & 1 - 2x^2 - 2z & 2yz + 2xw \\ 2xz + 2yw & 2yz - 2xw & 1 - 2x^2 - 2y^2 \end{pmatrix}$$

The problem can now be stated as follows. Suppose we are given the values of the elements of the rotation matrix R

$$R = \begin{pmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{pmatrix}$$

Our task is to recover the components of the corresponding quaternion q. Equating the above two forms of the matrix, following four expressions are generated

$$1 + m_{00} - m_{11} - m_{22} = 4x^2$$
$$m_{10} + m_{01} = 4xy$$
$$m_{20} + m_{02} = 4xz$$
$$m_{12} - m_{21} = 4xw$$

Forming a quaternion out of these components will generate $4x(x, y, z, w)$. The quaternion (x, y, z, w) was scaled by factor $4x$ and needs to be normalized and as a result we get the final quaternion [20].

In the end, the following data is exported to be further used by Blender to render the pieces: Image width, Image height, Focal Length, Camera translation vector and the quaternion. Intrinsic parameters along with $rvec$ and $tvec$ are used again in the later stage of the algorithm.

5.3 Creating a dataset of contours

The contour dataset is created by Blender. All of the pieces have been pre-modeled. The modeling process was quite straightforward. A compact camera was fixed, then all of the pieces were captured on the same spot and modeled by hand. All of the pieces have the same base which is reused in each model. Each

piece occupies its separate layer, which is activated when rendering that particular piece.

The data from the previous step of the algorithm are imported to Blender. The render output size is set according to the image size. Since rendering in the original resolution is expensive, the size of each dimension is reduced by a factor of 4. The focal length, the camera's location, and rotation are set, also from the input data.

A model of each piece is rendered at each of the 64 positions. These positions start from [-1.75, -1.75] and go to [1.75, 1.75], incremented at each step by 0.5 in a nested loop. These positions are set according to the previously described world scale. The only special case is the knight, which is not rotationally invariant. Knight is therefore rendered at each position 8 times, with each of these positions rotated on the vertical by 45 degrees.

The render is done via Blender internal rendering engine with a white background. The diffuse color of the piece is set to black with no specular, so the output is just a uniform black area representing the piece. Each render is then saved and used later.
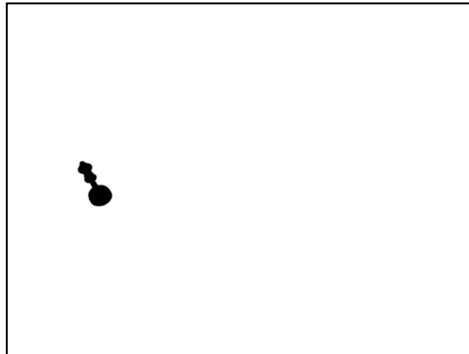
Figure 13: Queen rendered at position A2

The data created in Blender need to be extracted. Since each picture is essentially a black blob on a white surface, the left-most, right-most, top-most and bottom-most black pixels are calculated and their location is used to determine the region of interest for the piece. Then the piece is cut out from the render.

26

Figure 14: Queen cut out from the render

This time the edges are detected using mathematical morphology. It is a suitable approach for a black and white image. Since we are interested in one-pixel-wide edge, half-gradient by erosion is used [21].

where   represents half gradient by erosion of an image,   represents the input image and   represents the eroded input image. Each template is saved in 3 instances. One of the whole contour, one of the top half, one of the bottom half.

Figure 15: Left image is represent the contours of the queen. Top right image represents the top part of the contours of the queen and bottom right image the contours of the base. The base was actually taken from a pawn, since all of the pieces have the same base

5.4 Analysis of positions

Now that the chessboard is detected, the parameters of the camera are known and the templates for classification are ready, all that is left is to analyze the position. This includes the analysis of the chessboard, more notably its orientation resulting from the colors of the squares, a decision whether a square is occupied, and if so, what piece of what color is standing on it.

5.4.1 Chessboard color and orientation

The chessboard has squares of contrasting colors. Let's call them black and white. It is necessary to find the orientation of the chessboard. When the squares are detected, they are saved in a vector. This vector of squares starts with the left-most square on the picture and continues by the line away from the camera

Figure 16: Order of the squares saved in the vector of squares

The task is to find out, whether this left-most square is white or black. This information is also helpful for determining the occupation of squares. The configuration is determined using histogram analysis. Pixels contained in each of the squares are collected, transformed into a grayscale image and a histogram is calculated. The histogram is subsequently blurred and a global maximum is found. If this maximum is above a threshold, which in this case is the middle bin of the histogram, the square is classified as white, otherwise it is black.
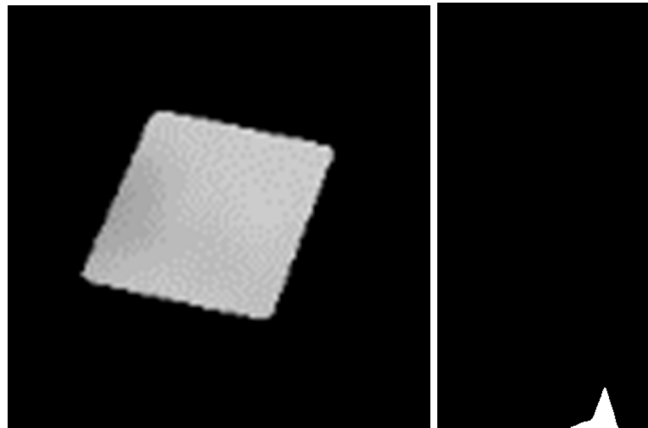
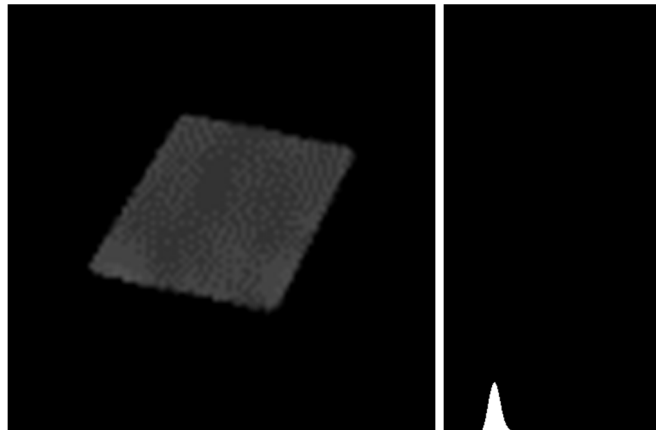Figure 17: A white square and its histogram.

Figure 18: A dark square and its histogram.

In many cases, the square color is classified incorrectly. A dark rook standing on a white square generates a histogram in which a global maximum might be below the threshold.
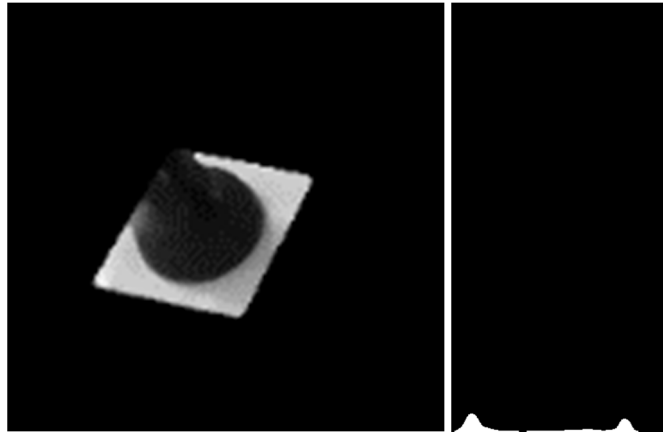
Figure 19: A white square with a dark piece on it. The global maximum of the histogram is in the dark spectrum and the square is therefore classified as dark.

This is not a problem as long as at least 33 out of 64 squares are classified correctly. Since there are 32 pieces in the beginning of the game, 32 squares are always free and these should be classified correctly, in the rest of the cases, only in some instances there would be a dark piece standing on a white square and vice versa, so the number of wrongly classified square colors is expected to be very low.

To combat this, two chessboard data structures are created. One starts with a white square, the other with black. Then the classified colors of the real chessboard are compared with these two ideal chessboards. Whichever of the ideal chessboard more closely resembles the classified colors is considered the configuration of the chessboard in the picture. So now it is known that when a square was classified as black, even though it is supposed to be white, it is probably occupied by a black piece. This kind of approach aids determining the occupation and the color of the pieces.

5.4.2   Occupation of the squares and color of the pieces

Two variants were tried to determine whether a square is occupied. Variant A exploits the fact, that all of the pieces have the same base. Variant B uses histogram analysis of the square and from the peaks in the histogram determines the occupancy of the square and the color of the piece.

Variant A:

The examined square is cut out from the image. Then the image cut-out (referred to as Cutout from now on) is blurred to get rid of noise and some imperfection of the chessboard that result in high gradient magnitude. Afterward, the Cutout is processed with the Canny edge detector. The edge map of the Cutout now contains a contour of whatever is occluding the square. It might be the base of a piece, a top of a piece standing at the neighboring square, nothing, or just some noise.

Figure 20:
Left Image: Base of a piece from the input photograph
Middle Image: Top of a piece from the input photograph
Right Image: Noise

To get rid of noise and less interesting contours, an area of each connected component in the edge map is calculated and components with areas below a threshold are removed. Since all of the bases are the same, the algorithm takes the contours of the bottom parts from the images rendered by Blender.

Figure 21: Bottom of a rendered piece

If there is a base of a piece standing on the square, it should be very similar to the contour of the bottom of a rendered piece. The bottom part template is searched for at every location of the edge map using template matching with cross-correlation. If the result is above a certain threshold, which is set according to the size of the cutout, the algorithm considers the square occupied by a base of a piece, therefore it is considered to be occupied.

The color of this piece is determined using a similar approach as with the classification of a color of the square. In this case, a histogram is made of only a few central pixels of the square. If the global maximum is above a threshold, the piece is considered to be light, otherwise, it is dark.

Variant B:

Variant B uses histogram analysis to determine occupancy and the color of the piece at the same time. After some deliberation, several cases of histogram shapes stood out as the dominant cases that can be found over the chessboard. There can either be an empty white square, an empty black square, a white square with a light piece, a white square with a dark piece, a black square with a light piece and a black square with a dark piece. All of these cases have various histogram types.

Figure 22:
Top left: an empty black square
Top middle: a black square with a dark piece
Top right: a black square with a light piece
Bottom left: an empty white square
Bottom middle: a white square with a dark piece
Bottom right: a white square with a light piece

In some cases, false positives and false negatives are likely to exist. These cases include a dark piece in bad lighting over a black square, giving the feel of uniformity and only one local maximum in the histogram.

Another case is when the top of a piece from neighboring square influences the histogram of an empty square. If, for example, a square is white and top of a dark rook from the neighboring square is visible, the histogram has two high peaks and the algorithm might classify it as occupied. This false positive is dealt with later. The peaks are found by smoothing the histogram with a kernel of a large size. Then onlyna few local maximas remain and their locations are determined. These maximas represent peaks.
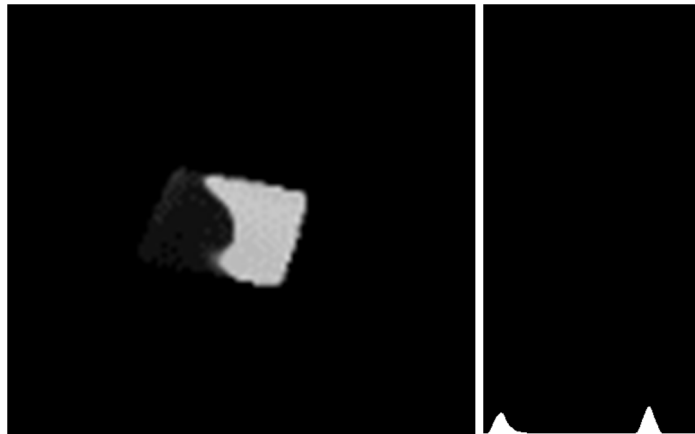
Figure 23: Large portion of a white square is occluded by a dark knight from a neighboring square

The white chessboard squares and the light pieces have both colors of quite different intensities, unlike black squares and dark pieces, where the intensity is almost the same. The first case is easier to solve. It is already known, whether the square is black or white. If the square is white, the histogram of that region will have a high peak in the region of the histogram with high intensities (to the right). If only this peak is present, the square is empty. If there are other peaks, the two highest are chosen. One of them should be at the right side of the histogram since the observed square is white. If the other peak is on the left side of the histogram, it indicates occupation by a dark piece. If it is in the middle of the histogram, in indicates occupation of a light piece. As a measure of filtering out false positives, the square is classified as full only if the second peak is above a threshold. This threshold depends on the size of the cutout. Top of a black bishop hovering over a square produces a peak on the left side of the histogram,

but since the top of a piece is thin, the peak would be low. Such square would be considered empty.

The dark squares are more complicated. To solve this problem, the histogram of the intensities in the square is smoothed with two different kernels, one small and one large. The small kernel has a size of 5 and the large kernel is three times as large. The histogram smoothed by the small kernel is rough, while the other is extremely smooth. The smooth histogram is observed. It should always have a peak on the left side as it is a black square. If a light piece occupies the square, there should be another peak in the middle of the histogram. If this is the case, and the light peak is above a threshold, the square is considered to be occupied by a light piece. The issue comes with determining the dark piece. We cannot say that the square is empty when the smooth histogram has only one peak on the left side because the dark piece and the black square have intensities too similar to one another. In this case, the algorithm looks into the rough histogram, where the peak is more jagged. Even though the dark piece and the dark square color are similar, unless the lighting is so unsatisfactory that they have virtually the same color in the picture, the area of the smoothed peak will have two local maximas in the rough histogram. If these two maximas exist, the square is considered to be occupied by a dark piece. In the case of aforementioned unsatisfactory lighting, the square is classified as empty and there is not much that can be done to correct this mistake, other than cross-reference it with analysis of photographs of the same position from different angles.

5.4.3   Classification of the piece

Each occupied square contains a piece that needs to be classified. This classification is done by template matching of the edge map with the contours of the templates created in Blender. First of all, a search region where the piece is standing needs to be specified and cut out from the photograph. It is not enough just to take the region of the square, now it is necessary to take area above the square, so the entire piece is present in the cutout. This is done by putting the projection matrix to use. We wish to extract a piece standing on a square. This piece fits in a box. The base of this box is the examined square and its height is the height of the highest piece, the king. This way even the largest piece can be

contained in it. For instance, a corner square would have a box with coordinates shown in Table 2.

Coordinate #	X (Blender units)	Y (Blender units)	Z (Blender units)
1	-1.75	-1.25	0
2	-1.75	-1.75	0
3	-1.25	-1.75	0
4	-1.25	-1.25	0
5	-1.75	-1.25	0.8
6	-1.75	-1.75	0.8
7	-1.25	-1.75	0.8
8	-1.25	-1.25	0.8

Table 2: Coordinates of a box containing a piece at the corner square of the chessboard

This box is then projected into picture space and represented with white color. A mask is thus created, the box is white and the rest is black. By overlaying the original image with the box, the result is a cutout of the square and the area above it where the piece is shown in its entirety.
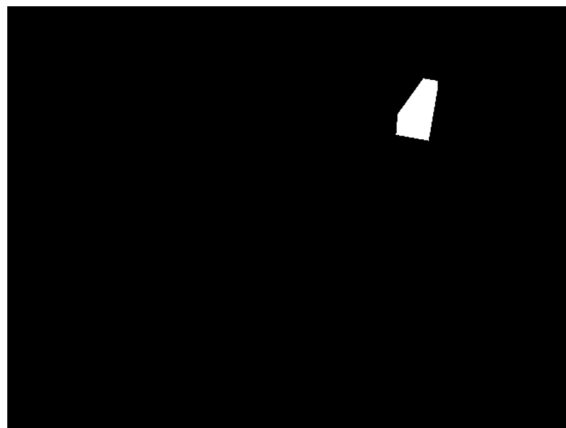
Figure 24: A mask overlaying a square

For each position, a vector   with 7 indexes is created and filled with a negative 1 at every position, except for the index 0. At that index, 0 is saved. Each index has a meaning, as demonstrated in Table 3.

Index #	Indicates
0	An empty square
1	A pawn
2	A bishop
3	A rook
4	A queen
5	A king
6	A knight

Table 3: Indexes representing each piece

If a square was detected as empty, the   is left as is. If it was detected as full, it is modified by the following procedure which is fairly similar to the variant A from the previous chapter. The cutout is processed by the Canny edge detector and template matching is used to determine the similarity of each template class to the piece in the cutout.

Figure 25: Contours of the piece segmented from the input image

The template of each piece from the specific square is laid onto every possible position in the cutout edge map. Each of these positions gives a response based on the cross-correlation equation. Each of the templates has one

position where it gives the highest response. The highest response for each template is saved at its respective positions in the vector. The index in the vector *Results* with the highest value signifies the class of the piece. If the square is empty, value at the index 0 is the highest, as the rest is -1. Due to the rounding errors when creating the template contours and cutout contours there is a possibility that the two will not be a perfect fit, only approximately very close to one another (for example while searching for a pawn and a pawn is in the cutout). The pawn from the template might miss the contours of the pawn in the cutout edge map by a margin of one pixel in many places, which would result in a bad response from cross-correlation. For this reason, the cutout edge map is blurred by a small Gaussian kernel of size 3 by 3. This is the general idea and 3 variations were tried out to get the classes.

Variant A:

This is the simplest variant. It is essentially the same procedure as the general idea described above. A position where a piece was detected is examined. A cutout is created. Canny edge detector is applied to the cutout. The cutout edge map is blurred. The pawn template rendered at the examined square is laid onto every single position in the cutout edge map and the cross-correlation equation is applied giving a response. The highest response for this particular class is saved into the vector *Results* at position 1.

Same procedure follows with the bishop, only this time the highest result is saved into *Results* at position 2 and so on. The only difference is the knight. It is not rotationally invariant, so for each square, it is rendered in 8 orientation. The maximum among all of these orientation is saved in the vector *Results* at position 6.

Variant B:

This variant uses two-step template matching. First, it tries to approximate the position of the piece and then calculates the final response at that position. The cutout contains the entire examined piece, as well as parts of pieces that are standing in the neighboring squares. These parts might be the tops of the pieces standing closer to the observer, or the bases of pieces standing further as observed in Figures 26 and 27.

Figure 26: Top of the knight's head is seen in the bishop's square.

Figure 27: Base of a piece from neighboring square is seen next to the knight's head

Since all of the pieces have the same base, which is quite large, while searching with the full template of each piece, the highest response for each piece template would be at a position where the template base is aligned with the base of the observed piece. This might prove to be problematic. If the bishop is the piece in the cutout, the bishop template is aligned with it and returns high response. A king is also aligned because the bases match and if there is some noise present, the top of the king's head might pass through some noise and return high response as well. Even though the cross-correlation is normalized, so the high pieces are penalized, the cross-correlation might return a king as the highest response in the _____ vector.

For this reason the first step of template matching is only with the top part of each template. In this case, the base will not play any role and the tops will be matched to the tops visible in the cutout. Since all of the tops are unique, only the top of the template of the observed piece should be matched to its correct position, the rest will be either scattered or matched to tops of the neighboring pieces that are visible in the cutout.

After the first step, each of the templates has an initial guess of where the piece is. This position is then used for comparison with the full template. In this case, the base makes a huge difference. The bases of templates whose tops did not match and are scattered all over the image would not be overlaid over a base

of the piece in the cutout and are heavily penalized. The bases of templates that match tops of the neighboring pieces have no bases visible in the picture, so once again, the bases do not match and the penalty is large. Only the template whose top was found as the top of the observed piece in the cutout is positioned correctly. Now the bases align and the response from cross-correlation at that position is large.

Figure 28: Top of the bishop was found in the blurred edge map of the cutout

Figure 29: Entire bishop is matched

Once again, the results of each template are saved in the   vector.

Variant C:
Similar to the variant B, this variant also estimates the position of each template by similarity measures. In this case, matching by phase correlation is applied. First, the template is fitted into an image of the same size as the cutout,

then the cutout edge map and the template are both blurred by Gaussians with the same kernel size and sigma. Then phase correlation is applied to estimate the relative translation offset between the blurred template and the blurred cutout edge map. Blurring was applied to provide more information to the template and the cutout edge map.

Figure 30: Heavily blurred edge map

Figure 31: Heavy burred template

Each template now has its estimated position. At this position, the cross-correlation between the original, non-blurred template and a slightly blurred edge map is calculated and the result is saved in its respective place in the vector.

Each square now has its own   vector, according to which the final classification is decided. First piece to be classified is the king. There are two kings of opposite color. We look into   vector of every square and

remember all of the squares where the king index has the highest value within the vector.

If no such square exists, all of the vectors are sorted by their values at the fifth index and the first king is picked as standing on the square represented by the *Results* vector with the highest value at index 5. The second king is picked on the square with the first highest value at index 5 with the opposite color of the piece.

If one such square exists, the piece standing on that square is considered to be a king. Then the rest of the vectors are sorted by their values at the fifth index. The second king is picked standing on a square occupied by a piece represented by the *Results* vector with the highest value on the index 5 with the color opposite to the color of the first king.

If two or more such squares exist, the first king is picked standing on the square with the highest value on index 5 among these squares. The second king is picked standing on the square with the highest value on index 5 occupied by a piece of opposite color.

The rest of the pieces are classified in a different fashion. Bishops will serve as an example. All of the *Results* vectors where the bishop index has the highest value are collected. They are then sorted by this index, from highest value to the lowest. Top 4 vectors are picked and these are considered pure bishops. The reason for choosing 4 is that there is very little chance of having more than 4 bishops on the board at the same time. When a piece is promoted into a bishop, it is usually at a later stage of the game, when many pieces are already missing off the board. The same procedure was applied to the rooks, knights, and queens. The pawns had the limit of 16, as there can be no more than 16 pawns on the board. The rest of the squares were classified based on the values in their *Results* vector cross-referenced with already classified pieces. If a square was not a pure bishop, rook, queen, knight or a pawn, the index of the highest value of this square is chosen. If there are already too many pure pieces of class signified by that index, the index with the second highest value is chosen and so on until the index with the lowest value. All of the remaining squares are classified as empty.

In order to battle the occlusion problem and to improve accuracy, the program can also work with 3 photographs of the same positions from different

angles. After the 3 photos are separately processed, the final position is determined by their averaging. Each position is examined in all of the three photographs. If at least two of them have the same piece standing on the square, the piece is considered to be the final piece. In case all of the three photographs vote for something different, the piece from the photograph with the highest response from the classification on the square is used as a result.

5.5 Output

The configuration of square colors and piece colors are known, so orientation so the chessboard can be determined. Since a white square in the back rank is always on the right side and it is a high chance, that most of the light and dark pieces are still at the halves of the chessboard where they started at the beginning of the game, the numbering of the chessboard can be assumed. It is assumed in such a way, that two orientations with white squares in the right corners are tried. For both of these, each light piece in the first 4 ranks of the chessboard gets a point and each dark piece in the other 4 ranks of the chessboard gets a point. The orientation with the most points wins and is considered final and the numbering starts accordingly.

There is much information missing as far as the game is concerned. The sequence of moves to get into this positions is unknown, we do not know who made the last move, whether there was any castling and so on. For this reason, an incomplete Forsyth–Edwards Notation [22] is used to describe current positions. The notation includes piece placement, active color, castling availability, en passant target square, halfmove clock and fullmove number. Only the piece placement is given in the output.

6 Evaluation

6.1 Chessboard detection evaluation

The described approach was chosen over the Harris corner detector. The main reason was that, while Harris corner detector is a perfectly viable option for an empty chessboard detection in the scene, in this case, the occlusion posed by the pieces might hide some corners. Due to this reason, it was decided to go with line detection algorithm and Hough transformation serves this purpose very well. Even in the case that parts of a line are occluded, the non-occluded parts still form a line and give a good response in the Hough space.

The problem with this approach was whenever there was a detection of lines which did not belong among the chessboard lines but they laid in such positions that the outlier filtering failed. These lines include the edge of the table if the spacing between the chessboard and the edge of the table was about 6 centimeters and whenever the pieces aligned in such manner, that pixels of their contours created a line with the support of enough pixels. If this line was parallel to one of the groups of lines, the filtering based on spacing sometimes failed, because it detected unnatural space and decided it might be the end of the chessboard.

Overall success rate over 252 images was 219 correctly detected chessboards. This amounts to 87% success rate. The images where the chessboard detection failed were often taken from very low angles.

Figure 32: Images taken from low angles

As seen in the right image, the angle is projecting many of the pieces over the borders of the squares. As a result, the lines made up by this borders don't have a distinctive peak in the Hough space and they fail to be detected.

6.2 Camera calibration evaluation

The calibration of the camera was by the most successful part of the algorithm. It heavily relies on correct detection of the chessboard, but once that is achieved, it works with 100% success rate. All of the verifications of the camera positions were done by rendering the virtual chessboard with the camera set up according to the calculated intrinsic and extrinsic data, which was represented as a plane of dimensions 4 blender units by 4 blender units with its center at [0, 0]. The render was then overlaid over the image with the chessboard and visually checked as demonstrated in Figures 33 and 34. Every single photograph verified fit extremely well.
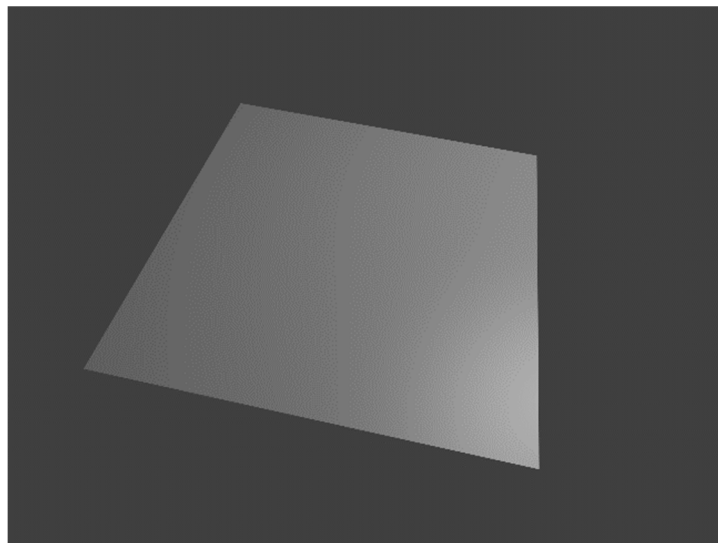
Figure 33: Plane representing the chessboard when rendered by a camera set according to the calculated intrinsic and extrinsic parameters
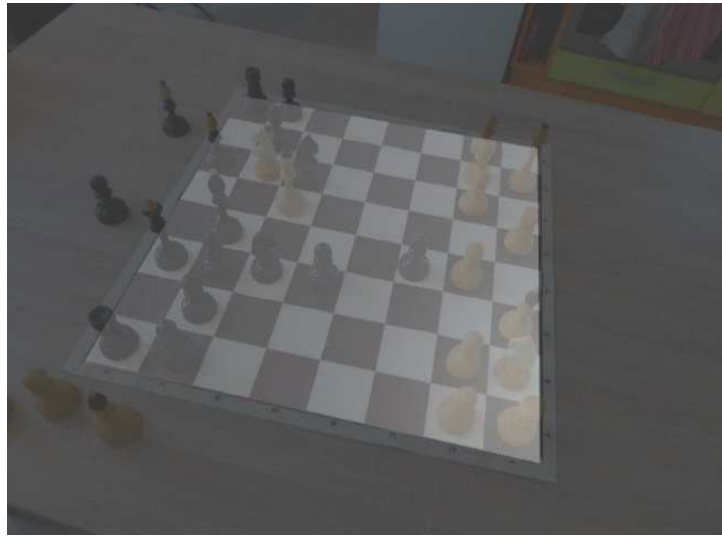
Figure 34: Overlay of the real chessboard with the generated chessboard plane.

6.3   Dataset of contours evaluation evaluation

This was a straightforward procedure that, if given correct input, returns the necessary output without any problems. The only decision-making process went into the method used for contour extraction. There were two options, either to use the Canny edge detector or mathematical morphology. Canny edge detector seemed to give visually less pleasing results. That was the reason to go with half-gradient by erosion.

6.4   Chessboard color and orientation evaluation evaluation

The output might be oriented in the wrong way if the game is in its advanced phase and for example, only the kings and one queen remain. These pieces can chase each other all over the chessboard so it is impossible to say at what side the dark and the light pieces started. Only the rule that the white square is on the right-hand side is correctly applied and then there is a 50-50 chance that the chessboard is oriented properly.

6.5 Occupation of the squares and color of the pieces evaluation

This was an important step in determining the chessboard positions and knowing, onto which squares to look and classify a piece standing on them. There were 2 variants tried, variant A with the matching of the base and variant B with the histogram analysis. Success rate of both of them was measured and it was decided that the histogram analysis is a more viable option.

The variant A was ran on 10 separate images, giving 640 squares to be analyzed. The analyzed results showed 77 wrong classifications of occupation, which amounts to 88% success rate. The variant B was ran on the same 10 images. The analyzed results showed 45 wrong classifications of occupation, which amounts to 93% success rate. The individual results can be seen in the table.

Image #	Variant A failed	Variant A success rate	Variant B failed	Variant B success rate
1	8	87.5%	6	90.6%
2	2	96.9%	4	93.7%
3	8	87.5%	7	89%
4	10	84.3%	4	93.7%
5	12	81.2%	6	90.6%
6	7	89%	6	90.6%
7	7	89%	5	92.2%
8	6	90.6%	1	98.4%
9	8	87.5%	5	92.2%
10	8	87.5%	1	98.4%

Table 4: Success rates of variants A and B

The suspected reason for failing with the variant A is the fact, that the visibility of the base is questionable. Whenever there is occlusion from a neighboring piece, the contours of the large part of the base would not be visible. In such case, the template matching would fail and the response would not be high enough to be classified as a base of the piece and the square thus remains classified as empty.

Another reason for false negatives is unsatisfactory lit scene. In the case that there is a dark piece standing over a black square, their colors might be so alike that no almost edges get detected resulting in template matching with low response. The square would be classified as empty.

The opposite case are the false positives. The algorithm works with edge detection and very often the edge detector finds unwanted edges which are long enough that they cannot be filtered out given their size. In such cases, while doing the template matching, the contour of the template base would get high responses when laid over such unwanted edges. These responses result in classifying the square as occupied.

These shortcomings are considered so deal-breaking that the variant A is not an option for a successful classification tool.

The variant B has its own caveats. False positives stem from a projection of the top of some piece onto an empty square. Here is an example of such issue
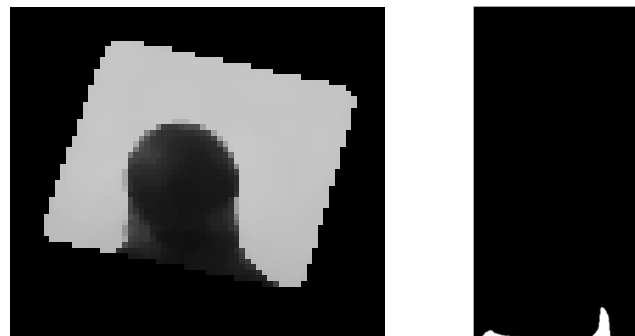
Figure 35: Top of the pawn visible over an empty square and the respective histogram

Even though there is a limit to the size of a peak to be considered important, the peak representing the color of the neighboring piece is so high that the square must be classified as occupied.

Another reason for false positives is once again strong lighting hitting the imperfections of the chessboard. This might result in an uneven histogram with important peaks all over the intensity spectrum. In such case, when there is a high peak over the low intensities and a high peak over the high intensities, the square is classified as occupied. For this reason, it is important to be vary of the

environment and try to take such images from such angles, that the source light is not reflected by the chessboard and does not create a high specular area.

Whenever false negatives arise, it is mostly by having a dark piece over a black square. In a dark environment, such colors tend to make a homogenous region and the object cannot be recognized.

The color of the pieces is done by the method tied to the variant B, as a side product of the histogram analysis. Out of 218 pieces on chessboards in the 10 images, 226 were classified correctly which amounts to 96.5% success rate.

6.6 Classification of the piece evaluation

The classification success rate is decided by the number of squares that were classified fully correctly, with respect to the occupation, piece type, and its color. The classified pieces went through 3 variants. Variant A was the simplest and as expected returned the poorest results, barely getting over 80% success rate.

Variant B implemented 2 step template matching. A sample result of the algorithm for the photograph in Figure 36 is demonstrated in Figure 37.

Figure 36: Sample position
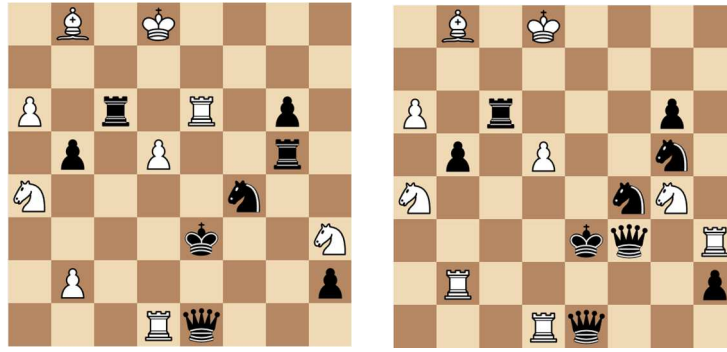
Figure 37:
Image 1 on the left shows actual positions
Image 2 on the right shows calculated position by the algorithm using variant B

Figure 38 shows the same position from different angles and Figure 39 shows the results of the program from the photographs in Figure 38.

Figure 38: Same positions from different angles
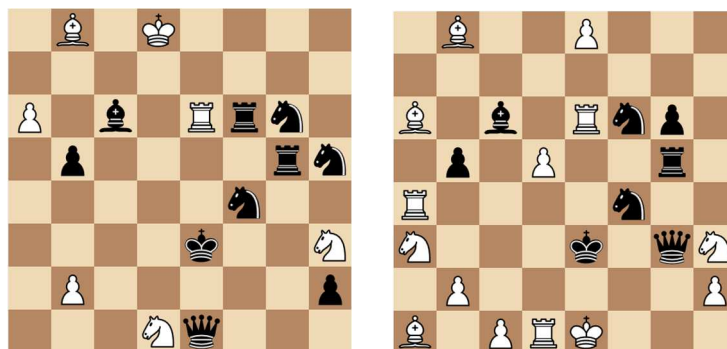
Figure 39
Image 3 on the left calculated position by the algorithm using variant B from the left image in Figure 38
Image 4 on the right calculated position by the algorithm using variant B from the right image in Figure 38

Success rate demonstrated by image 2 in Figure37 is 89%, success rate demonstrated by image 3 in Figure 39 is 90.6% and success rate demonstrated by image 4 in Figure 39 is 81.3%. If all of these results from the same positions are combined, much higher success rate is achieved. This is given by the fact that from some angles some pieces are occluded and the classification yields wrong results whereas from the other angles they are classified correctly. The same applies to the square occupation where averaging results from several photographs gives more precise results, as some of the photographs might have occlusion and lighting problems and the occupation is decided by histogram analysis. The averaged result is shown in the Figure 40, with the real positions in the left image and the averaged positions in the right image.
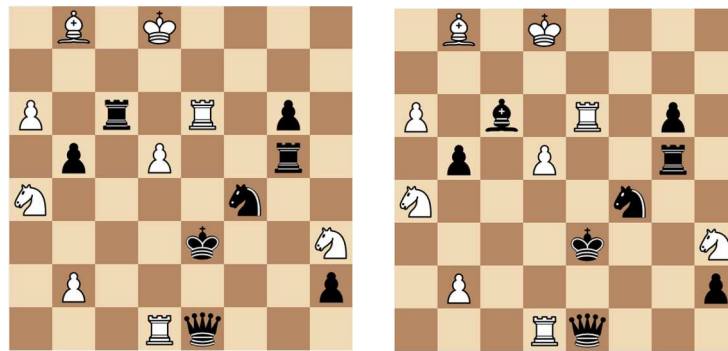
Figure 40:
Image 5 on the left shows actual positions
Image 6 on the right shows averaged position from 3 photographs

The success rate for combined results for variant B is 98.4%. Several more datasets were tried and the combined result for three images was a bit lower, on average performing at 93.58%. These results can be observed in Table 5 in Attachment A.

The same procedure was applied to variant C with less success. On the same dataset, it performed with an average rate of 81.3% on single image whereas combined success rate from 3 images was 87.5%. The overall success rate is

much lower than the variant B and it was therefore decided to keep the variant B.

There were several types of errors that showed while testing the classification. The errors can mostly be contributed to classifying a piece wrongly due to issues with similarity with another piece, and also to classifying piece as a knight, because of 8 knight templates.

The similarity issues were mostly visible between the pawn and the rook, the queen and the king, and the bishop and the king. The pawn and rook have approximately the same heights and when viewed from a higher angle, the contours of the thin pawn neck are mess visible so at the end both pieces appear as short lumps. The queen and the king similarity was going to be an issue from the beginning. The top of the queen is a little different but due to some rounding errors when matching templates, a king might appear as a queen and vice versa. Then during the analysis of the Result vectors, when too many kings were found, the top two might actually include a queen with high response to the king template and a queen is analyzed as a king. Later a piece that was actually a king but has a lower cross-correlation response than the others can no longer be classified as a king, as both kings have already been decided. This piece might become a queen and in the end we are left with switched king and queen. The bishop, when looked from above is also very similar to a king and the same issue applies to this case.

The problem with knights is the number of templates that need to be tried. There are more knight templates than the sum of other types of pieces. It is, therefore, possible that one of the knight templates will be rotated in such a way that it returns a high response. Even though a limit on knight was posed when classifying the Result vectors, the highest response might be given by a piece that is not actually a knight.

Another issue that surfaced with averaging 3 photographs was matching the orientation of the chessboards. In some situations, 2 chessboards are oriented one way and 1 chessboard is oriented by 180 degrees. This is due to the fact, that voting for the orientation is based on colors of the pieces, trying to have dark pieces towards the 8th rank and light pieces towards the 1st rank. Sometimes an occupancy or a color of a piece might be misclassified and in unfortunate cases it

influences voting for orientation of the chessboard so much, that it gets rotated by 180 degrees.

6.7   Performance evaluation

The program can be split into 3 phases, depending on the task they are executing. Phase 1 consists of chessboard detection and camera calibration. Phase 2 is rendering of the templates. Phase 3 deals with occupancy of squares, colors and piece classifications.

The average time to perform Phase 1 is 23 seconds. In the scenario that the chessboard does not get detected, the program tries a few different parameters for h-convex transform and parameter filtering. Each time this is done, it takes about 23 seconds and if after a few tries it does not detect the chessboard correctly, it fails. Having to repeat the chessboard detection is not too common of an occurrence.

The average time to perform rendering is 90 seconds. There are 512 renders, each of them takes just a fraction of a second to be rendered, but the saving of each render is costly.

The average time to perform the classification is 45 seconds. It varies depending on how many squares are classified as empty, as there is no need to do piece classification on these squares.

7 Conclusion

The program for chess position recognition from a photo proposed in this thesis has its benefits and caveats. It tries to classify the pieces using classical image processing tools and avoids machine learning. With this approach, there is no need to create a dataset of images, only to have their models that can be generated very quickly.

In this work, the models were created by hand. It took very little time but it has downsides, such as a problem with accuracy. The pieces were modeled according to an image which itself might be distorted, so the models might suffer from false geometry. To combat this I propose having the pieces generated with photogrammetry software in further works. This kinds of software take the pieces photographed from a few angles and return highly accurate 3D mesh of the object. This automatized approach might speed up the modeling process.

The rendering is currently done via Blender internal engine because of its easy use. It suffers from speed issues, so all of the pieces had to be rendered at ¼ of the sizes in the photograph, which had to also be subsequently scaled to ¼ of its size. Using openGL and rendering on GPU, while not needing to save and load each frame but to work with it right away could be a large time boost. Due to complexity of the program, I had decided not to try this approach but rather focus on piece classification.

As of now, the piece classification is the part most likely to fail. As described before, if the image is taken with enough care, so it is not blurry and the angle is chosen so the pieces do not cover the grid, the chessboard detection works well. The piece generation via Blender has 100% success rate as it is a fairly straightforward process and the calculation of the camera parameters is non-problematic.

The classification, on the other hand, is sort of a guessing game. Occlusion posed by pieces is very troublesome and with this program seems impossible to deal with from one photograph. When a king is occluding a pawn, the square with the pawn can be classified as any piece, because the template of pawn itself
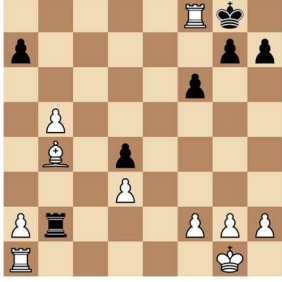
will not find the occluded pawn and any random template might have the highest response. For this reason, I propose that the classification is done from at least 3 photographs from different angles and a piece on the square is voted for by classifications of that square form each photograph.

8 References

[1]   KRAUTHAMMER C. *Be Afraid [online].* 1997 URL: <https://www.weeklystandard.com/be-afraid/article/9802> [cit. 15-05-2018]

[2]   *About Chess.com [online].* URL: <https://www.chess.com/about> [cit. 12-05-2018]

[3]   HARRIS C.; STEPHENS M. "*A combined corner and edge detector.*" Proceedings of the 4th Alvey Vision Conference. pp. 147-151 (1988)

[4]   DUDA R.; HART P. "*Use of the Hough transformation to detect lines and curves in pictures,*" Comm. ACM, vol. 15, pp. 11-15 (1972)

[5]   ESCALERA A.; ARMINGOL J. M. " *Automatic Chessboard Detection for Intrinsic and Extrinsic Camera Parameter Calibration,*" Sensors, 2010

[6]   SONKA, Milan., Vaclav. HLAVAC a Roger BOYLE. *Image processing, analysis, and machine vision*. 3rd ed. Toronto: Thompson Learning, c2008. p. 215 ISBN 0-495-08252-x.

[7]   *Camera Calibration [online].* URL: <https://docs.opencv.org/3.1.0/dc/dbb/tutorial_py_calibration.html> [cit. 30-04-2018]

[8]   *Camera Calibration and 3D Reconstruction, section SolvePnP* [online] URL: <https://docs.opencv.org/3.4.0/d9/d0c/group__calib3d.html#ga549c2075fac14829ff4a58bc931c033d> Last updated on Dec 22, 2017. [cit. 2018-25-04]

[9]   SONKA, Milan., Vaclav. HLAVAC a Roger BOYLE. *Image processing, analysis, and machine vision*. 3rd ed. Toronto: Thompson Learning, c2008. p. 561-566 ISBN 0-495-08252-x.

[10]   DANNER C; KAFAFY M. (2015).*Visual Chess* Recognition [Online]. Available: <https://web.stanford.edu/class/ee368/Project_Spring_1415/Reports/Danner_Kafafy.pdf> [cit. 2018-04-26]

[11]   DING J. *ChessVision:Chess Board and Piece Recognition [Online].* Available: <https://web.stanford.edu/class/cs231a/prev_projects_2016/CS_231A_Final_Report.pdf> [cit. 2018-04-26]

[12] SUKRITGUPTA17 *Chess-Board-Recognition [online]. Available:* <https://github.com/SukritGupta17/Chess-Board-Recognition> Last updated on Aug 31, 2017. [cit. 2018-05-17]

[13] TAKITA K.; AOKI T.; SASAKI Y.; HIGUCHI T.; KOBAYACHI K. (2003) *High-accuracy subpixel image registration based on phase-only correlation.* IEICE Trans Fundam Electron Commun Comput Sci E86A(8):1925–1934.

[14] *Phase correlation [online]* URL: <https://en.wikipedia.org/wiki/Phase_correlation> Last updated on Jan 10, 2018. [cit. 2018-05-02]

[15] *Laws of Chess: For competitions starting from 1 July 2014 till 30 June 2017 [online]* URL: <https://www.fide.com/fide/handbook.html?id=171&view=article> [cit. 2018-05-06]

[16] *ČESKÁ KLUBOVKA krása šachových figurek [online]* URL: <http://sachy.csla.cz/> [cit. 2018-05-06]

[17] *EXIF [online]* URL: <http://www.exif.org/> [cit. 2018-05-06]

[18] SOILLE, Pierre. *Morphological image analysis: principles and applications.* 2nd ed. New York: Springer, c2003. p. 203 ISBN 3540429883.

[19] *Camera Calibration and 3D Reconstruction, section Rodrigues [online]* URL: <https://docs.opencv.org/3.4.0/d9/d0c/group__calib3d.html#ga61585db663d9da06b68e70cfbf6a1eac> Last updated on Dec 22, 2017. [cit. 2018-04-25]

[20] DAY M. (2016). *Converting a Rotation Matrix to Quaterion [Online].* Available: < https://d3cw3dd2w32x2b.cloudfront.net/wp-content/uploads/2015/01/matrix-to-quat.pdf> [cit. 2018-04-26]

[21] SOILLE, Pierre. *Morphological image analysis: principles and applications.* 2nd ed. New York: Springer, c2003. p. 86 ISBN 3540429883.

[22] *Forsyth–Edwards Notation [online]* URL: <https://en.wikipedia.org/wiki/Forsyth%E2%80%93Edwards_Notation> Last updated on Jul 21, 2017. [cit. 2018-05-02]

[23] MAYANKLAHIRI *easyexif [online]. Available:* <https://github.com/mayanklahiri/easyexif> Last updated on Aug 11, 2017. [cit. 2018-05-12]

[24] *Blender2.79 [online] URL*: <www.blender.org>

A Attachment

Real position	Calculated position by averaging 3 photos	# of wrongly classified positions	Success rate
		7	89.0625%
		3	95.3125%
		4	93.75%
		7	89.0625%
		6	90.625%
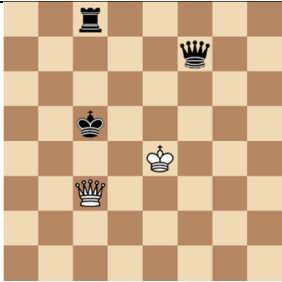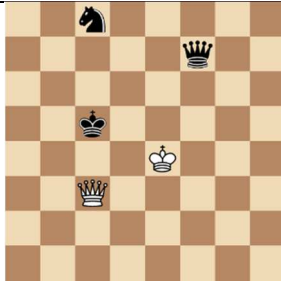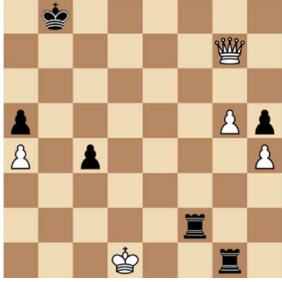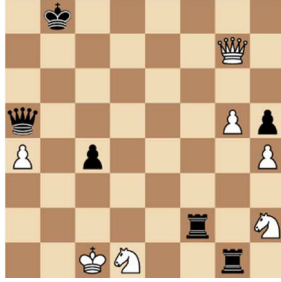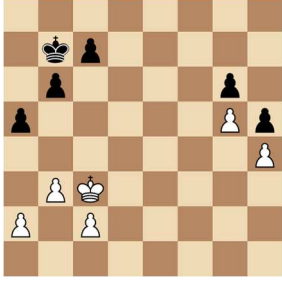
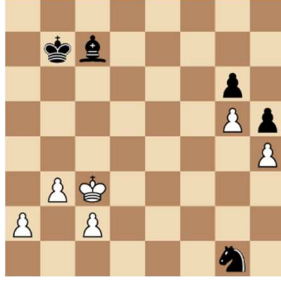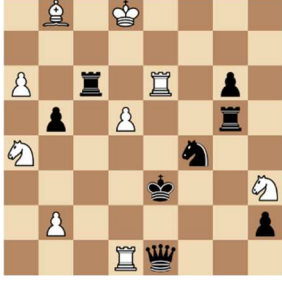|			
		1	98.4375%
		4	93.75%
		4	93.75%
		1	98.4375%

Table 5: Success rate of variant B with averaging 3 images

B Attachment

B.1 Technology

The program was created using C++ programing language, along with the OpenCV and EasyExif [23] libraries. OpenCV is used for image processing and EasyExif for parsing the EXIF information from photographs.

Rendering part is done in Blender [24]. It uses Blender internal render engine along with bundled python scripting tools for setting up the camera and the scene. The program currently works only on Linux systems.

B.2 Instruction for running the program

The program utilizes OpenCV functions. In order to compile it, OpenCV needs to be installed. Blender is bundled along with the source files. Unpack the zip file containing source code, data and Blender. Open the terminal and navigate into the unpacked folder. To compile the program call `./compile.sh.` It might be necessary to change permissions for the file, use `chmod 777 compile.sh`

In case there is an issue with OpenCV, check the path, whether it is valid. To run the application, call `./binary x y a b c` where `x` is sensor width, `y` is sensor height, and `a, b, c` are the paths to the three images of the same position. In case only one image needs to be analyzed, `b` and `c` can be omitted.

There are test images in the folder Images. The sensor size must be set according to the device used to capture the images. The folder Images has two subfolders, iPhone and Panasonic. The folder structure is organized, so the photos of the same position are in the same folder. The images contained in them should have sensor size set according to Table 6.

Sensor Size	iPhone	Panasonic
x	4.89	6.16
y	3.67	4.62

Table 6: Sizes of sensors

B.3 Output

The output is returned in the terminal, as well as in text files. When computing the position from one photograph, the result can be found in **results.txt**. When computing the position from three photographs, the individual results from each image can be found in **results.txt** and the averaged result is outputted into **AverageResult.txt**