Chess Vision

Final project report

Ben-Gurion University of the Negev
Faculty of Engineering Science
Dept. of Electrical and Computer Engineering
Introduction to Digital Image Processing 361-1-4751

Fall 2019

Maor Assayag	318550746
Eyal Zuckerman	208416099
Refahel Shetrit	204654891
Yaniv Okavi	315521021

אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

Table of contents

אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

Abstract

Throughout history chess has fascinated millions of people around the world with the complexity it presents, not only as a hobby but also as an opportunity to demonstrate creativity and improve calculated decision-making.

Over the years, the task of electronically tracking a game of chess has been undertaken by many people. A digital chess set can easily record a game automatically, but the specialized chessboard and pieces can be costly.

While more technically challenging, the use of image processing to detect and identify a chessboard and the configuration of its pieces avoids the need for a digital chess set. Furthermore, image-based detection of chess pieces is a vital step in building chess-playing physical machines. Such machines can be used for fun or research and have been considered as an interactive toy that helps in developing the learning abilities of chess.

A goal of this project is responsive identification of both the board and pieces using image processing. This research's [1] [2] and many more provide some applicable approaches to identifying 3D chess pieces using Convolutional Neural Network (CNN) - due to insufficient results from more robust image-processing algorithm (e.g. SURF/SIFT).

For the goal of using those image-processing tools, the using of 3D pieces was ruled out. We decided to approach the piece identification part of the problem by using 2D pieces, which is widely used for learning chess.

In addition to tracking the state of a chess game, the system will provide a visual presentation of the legality of the move played and suggestion for the best next move using a chess engine.

This project contains several steps involving image-processing tools such as transformations, color spaces, segmentations etc.'

Keywords: chess, image-processing, SURF, transformation, classification, hand-recognition

System design

Setup

The system will consist of a physical chess board with 2D chess pieces and above it a regular camera at an overhead angle connected to a laptop.

the camera will also capture a hand gesture (number from 1-4) represent the desired engine level of the suggesting moves.

The camera is set to have a 1280x800 resolution 10 frames per seconds (default FPS).

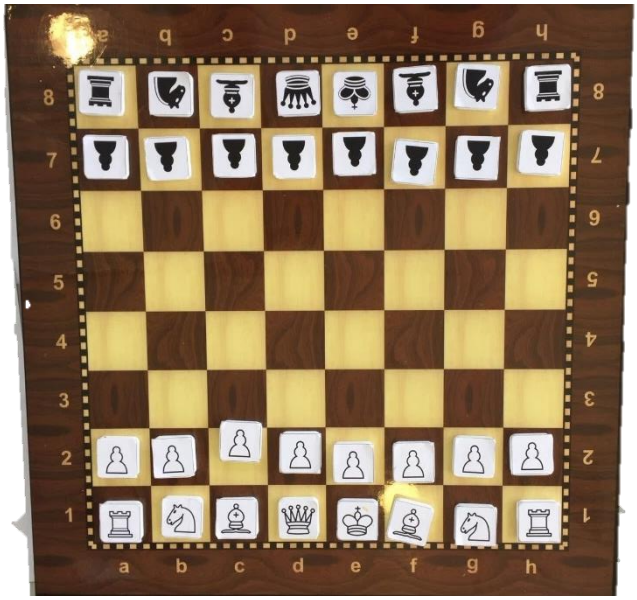


Figure 1.1 – setup example

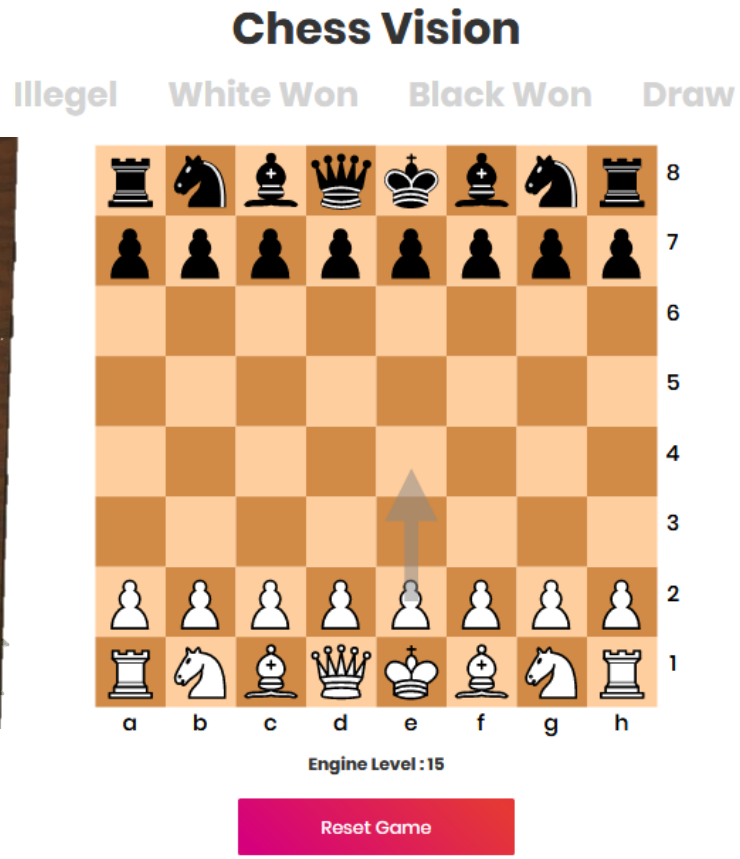


Figure 1.2 – Web app UI

Development frameworks

The image processing part (hand gestures recognition, chessboard segmentation, chess pieces classification and calibration logic) is done using MATLAB R2018 with the image-processing tool-kit.

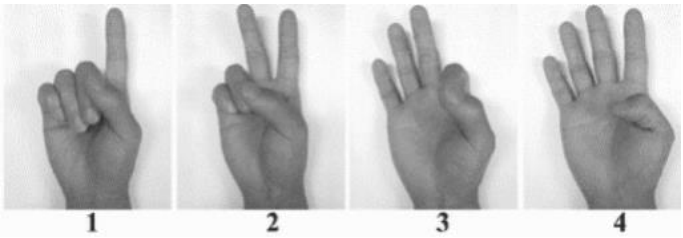
The MATLAB code is communicating with a Python code via a text-based communication (read & write from several text files). In general, the MATLAB side is responsible for delivering the detected move done by the player.

The state of the chess game is mainly tracked by the Python side, which consist of multithreaded code that handle a microframework (web app server for visualization) thread & chess game tracking with best move suggesting feedback (and legality flags etc.') thread.

The microframework is designed using FLASK web with SocketIO which gives the Flask web application access to low latency bi-directional communications between the clients and the server – that enables us to update the local webpage with the current state of the games & mandatory visualization for the user (using jQuery that updates the HTML code of the web page).

Features

1. Providing a visual presentation of suggestions for the next best move using microframework (a web app) maintained via python which entering the current state of the board to a chess engine respondent feedback with the best move for the player whose turn to play & a legality feedback of the last turn using a chess python library.
2. Responsive system which give immediate feedback required to support fast-phase chess games.
3. Hand movement interrupt the view angle of the camera will not disrupt the system analysis
4. The level (ELO points system) of analysis by the chess engine will be determine with hand-gesture (1 to 4) image recognition in the start of the system.

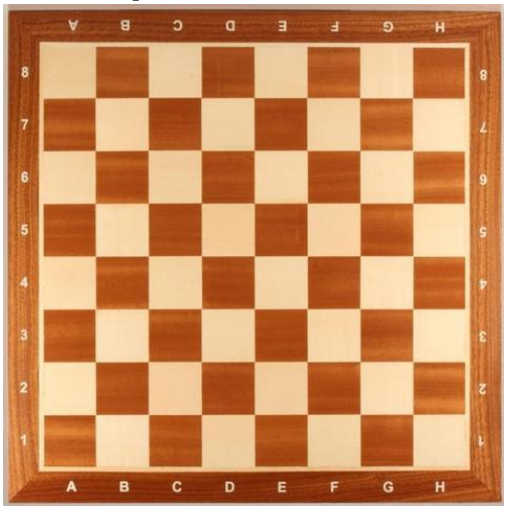


5. Easy to understand GUI (General user interface) which include the move suggestions in a clear way.
6. High level of accuracy regarding angles recognition of the chess board pieces.
7. Starting a game with any legal position, which detected by image-processing and converted to FEN string (will be explained later) for the Python side.
8. Dynamic coding that enabled using the Web app for any future use (e.g. tracking a game with special sensors, robotic arms etc.) with a simple communication pipe (text file based).

Assumptions

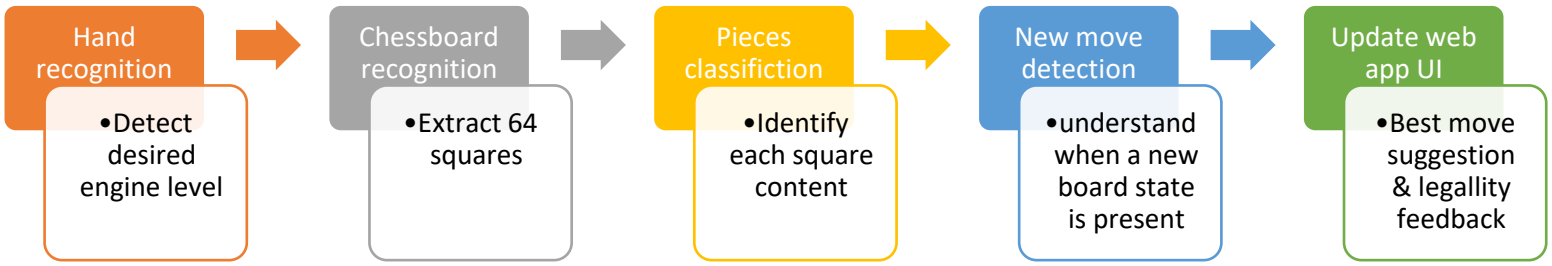
1. An environment with reasonable lighting conditions suitable for a standard camera quality live video feed (normal room lighting).
2. The camera is setup to be with overhead angle of the physical chess board.
3. The pieces consist of the same 2D graphics that our algorithm data set is trained with (we will elaborate on this later in the pieces classification part at page 34).

4. The background for the hand-gesture recognition camera setup is preferred to be a color that is uniform and clear.
5. Pre-installed python environment (page 58) and MATLAB with the image-process tool-kit & USB camera with the camera add-on in MATLAB.
6. Simple flat chess board – the main assumption is that it's not wavy, For example



System Flow

1. Waiting for reset button on the UI to be pressed.
2. Recognize when a hand entering a camera live feed, then recognize the number that the hand gesture applies. This number (1-4) will be convert to the range 1-20 for the engine level of analysis and suggestions.
3. Chessboard recognition - The objective regarding processing the chessboard is to find a mapping from the original image to an ideal chessboard, e.g. extract each 64 squares.
4. Chess pieces classification – At the beginning of a match each of the 64 squares will be classified to perform a starting position detection which enable us to create a special string called FEN for the chess engine to understand the starting position of the game.
5. For now-on, an algorithm of pieces movement detection will take place to determine the move from-and-to square location.
6. The starting position and each new move will be sent via text-file-based communication to a python code for web app updating & best move/illegal move feedback on the web page UI.
7. If the user done illegal move, until the last legal position is restored on the board the user will get a feedback of illegal moves (MATLAB).



Analysis

In this section we will explain each step on each stage of the system, from the algorithms behind them, why we choose those algorithm & analysis of the results, pros & cons etc.

Hand gestures recognition – Engine level setup

Background

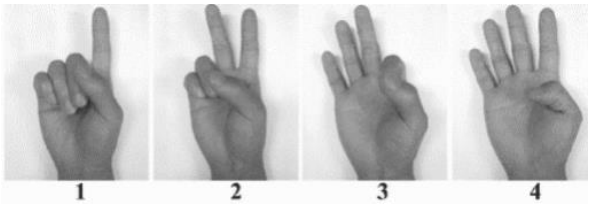
The Elo rating system is a method for calculating the relative skill levels of players in games such as chess. A player's Elo rating is represented by a number which increases or decreases depending on the outcome of games between rated players.

Chess engine rating aim to provide statistically significant measures of relative engine strength.

If we want to create a platform which give suggestion to players, we want to be able to control the level of the engine analysis and give a relatively close strength of ideas and understanding of chess as the player strength at chess. If we tried to recommend ideas of play to a starting player that seems out of the world, this will only bring frustration instead of help with the learning curve and understanding the game.

We could easily place the choice of the engine level in the web page UI, but this stage is done with image processing by instruction.

In this part we are performing hand gesture recognition to determine what number the player choose (1-4). This number will be sent via text-file-based communication (see page 50), which then will feed into the engine setup option as the engine level strength of analysis for best next move suggestions.



Algorithm flow

Live feed of images

When we start the system, the camera that responsible for this part is capturing images at the rate of image per 0.2 seconds. On each capture image, we calculate the difference between the current and the initial image. By trial and error, the thresh hold for a moving hand to enter the scene is at least max(new_image-initial_image(:)) > 120 – this threshold can change in varies lighting conditions (but not drastically). (More on this tech in the new move part, at page 27)

When this step recognizes a new object in the scene, we procced to the gesture detection.

Figure 2.1 – example of a captured image

Its also worth noting that we are using only 1 camera for both chess board tracking and gesture at 1280x800 resolution, which enable us to dedicate a constant section in the frame for this operation (using image cropping *imcrop*).

Skin detection – binary masking

Skin detection is the process of finding skin-colored pixels and regions in an image or a video.

The primary key for skin recognition from an image is the skin color. But color cannot be the only deciding factor due to the variation in skin tone according to different races.

One simple method is to check if each skin pixel falls into a defined color range or values in some coordinates of a color space.

The following factors should be considered for determining the threshold range:

- Effect of illumination depending on the surroundings.
- Individual characteristics such as age, sex and body parts.
- Varying skin tone with respect to different races.
- Other factors such as background colors, shadows and motion blur.

The YCbCr color space is widely used for digital video. In this format, luminance information is stored as a single component (Y), and chrominance information is stored as two color-difference components (Cb and Cr).
Cb / Cr represents the difference between the blue /red component and a reference value.

We decided to work in this color space for the skin detection following the suggestion of a research that tested in depth this task and the use of a difference color spaces [3]. For simplicity sake, the threshold has been taken from this simple function [4] for creating the mask of the hand. After we played more with the numbers, we decided not to change those values that seems to work well.

From a simple thresh hold in the YCbCr color space we can get an initial hand mask. From checking where is most of the mask is we can rotate the picture such that the hand will be on right side.

אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

Figure 2.2 – rotated hand mask

Cleaning the mask

First, we are cleaning any noise from the picture by filtering the objects with *bwareaopen(mask,10000)* which removes all connected components (objects) that have fewer than 10000 pixels from the binary image BW, producing another binary image.

Figure 2.3 – filtered hand mask

אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

With *imclose(I,SE)* we perform morphological closing on the binary image I, returning the closed image, J. SE is a single structuring element object returned by the function *strel('disk',radi),* when *radi* is a radius calculated with the image dimension and the weight of the image(the percentage of the mask). The morphological close operation is a dilation followed by an erosion, using the same structuring element for both operations.

For making sure that any holes have been filled, we are running *imfill(mask, 'holes')* that fill holes in the input binary image *mask*. In this syntax, a hole is a set of background pixels that cannot be reached by filling in the background from the edge of the image.

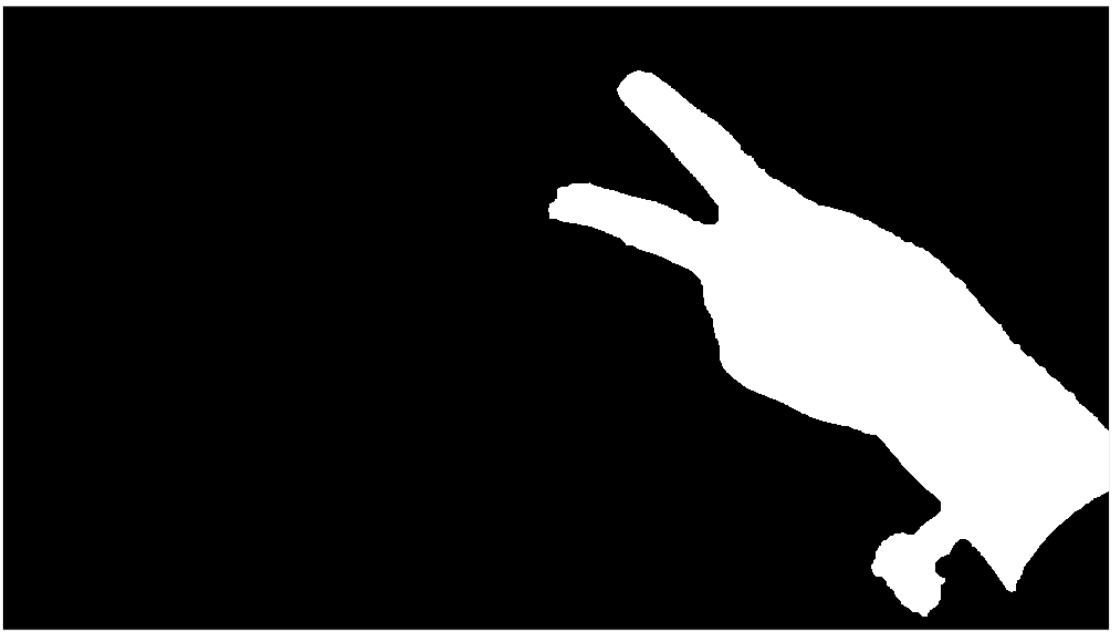


Figure 2.4 – Clean hand mask

Counting the fingers

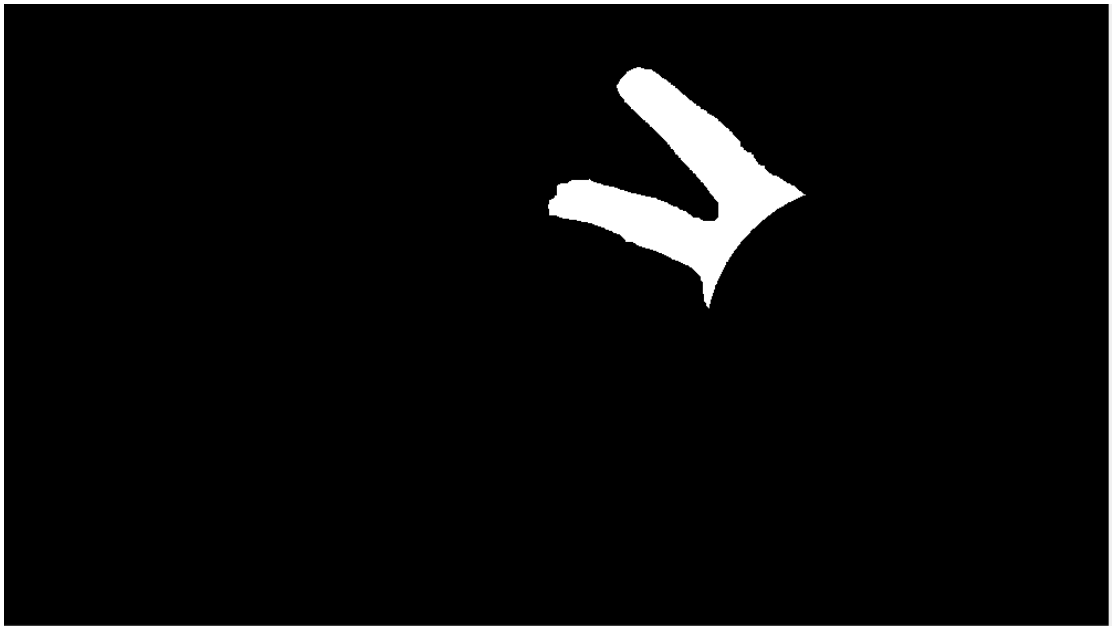
The idea is to multiply a black circle in the center-mass of the hand & a circle at the right bottom corner (erase the wrist) with the mask and increase the radius until we are left with unconnected parts of fingers in mind that we can easily find how much connected object is in the mask in any change of the radius.

אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

To find the center-mass of the binary mask we are using *regionprops(BW, properties)* that returns measurements for the set of properties specified by properties for each 8-connected component (object) in the binary image, BW. In our case we want the center of mass of the region. So, we choose the property *'centroid'*.

Figure 2.5 – Center of mass

Then. We are removing a cycle shape around the center of mass in an increasing radius, which eventually trim the binary mask to contain an un connected finger parts, as follow :



אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

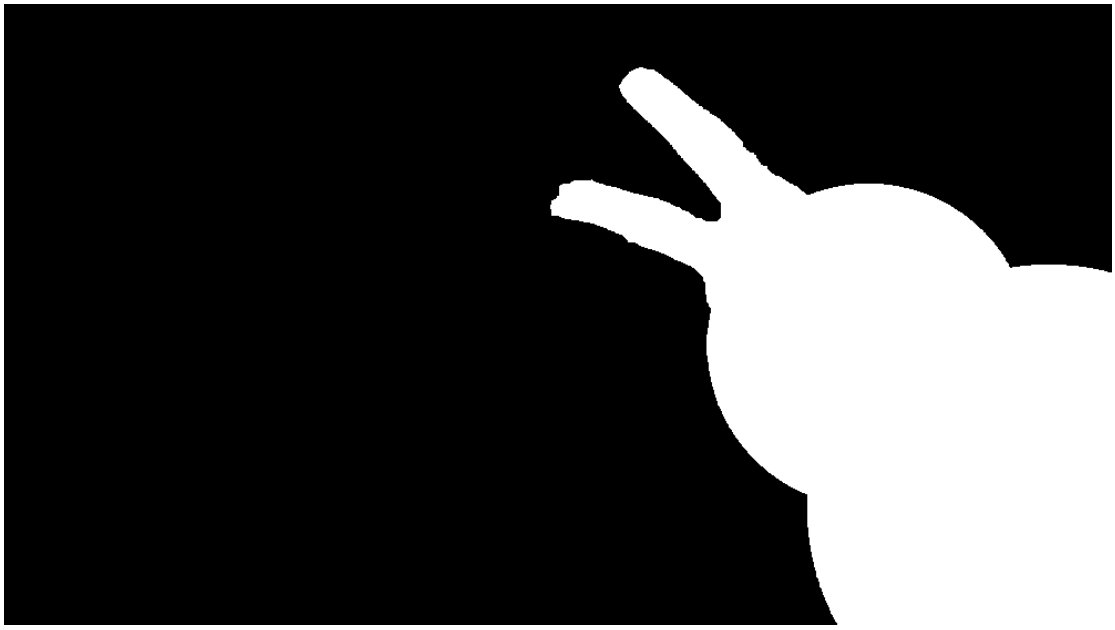


Figure 2.6 – highlight of the 2 cycles

As we can see, the finger is not yet unconnected object. The next radius will cut off their connection point, as follow:

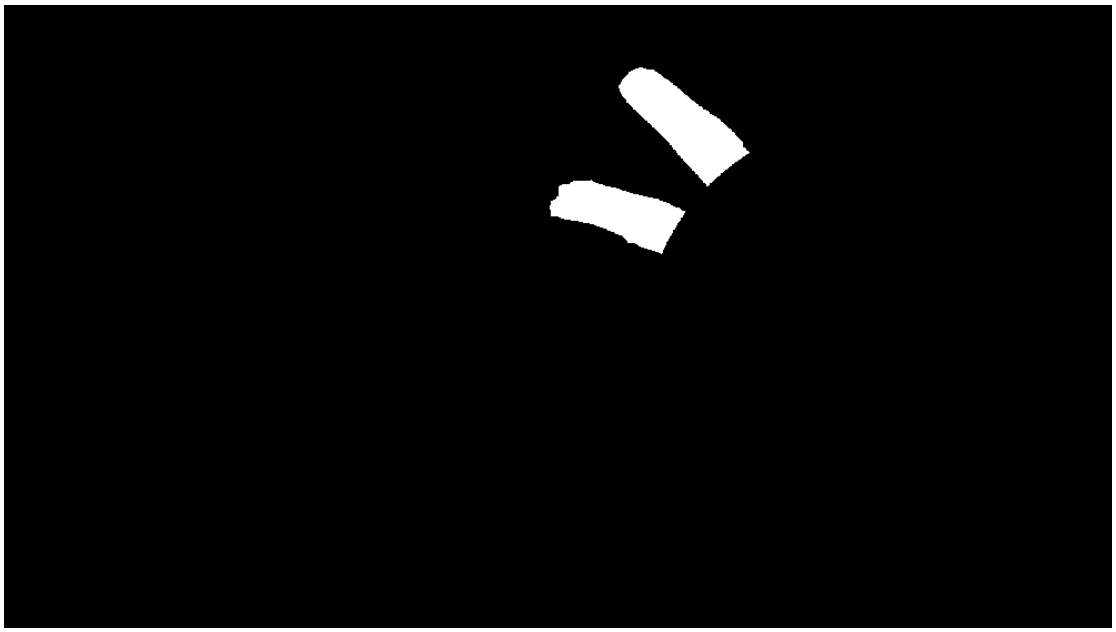


Figure 2.7 – Now the fingers are 2 separate uniform objects

For each radius increase we are checking how much connected components is there in the binary mask using the function *'bwlabel'*.

In the final step, we are checking for the max median number of connected components of those binary mask variations (after alienation of the 15% of the edges samples), which are more likely to represent the number of fingers in the hand gesture that have been captured.

```
n =

     2
```

Testing

Examples of 1-4

Figure 2.8 – original image of 3

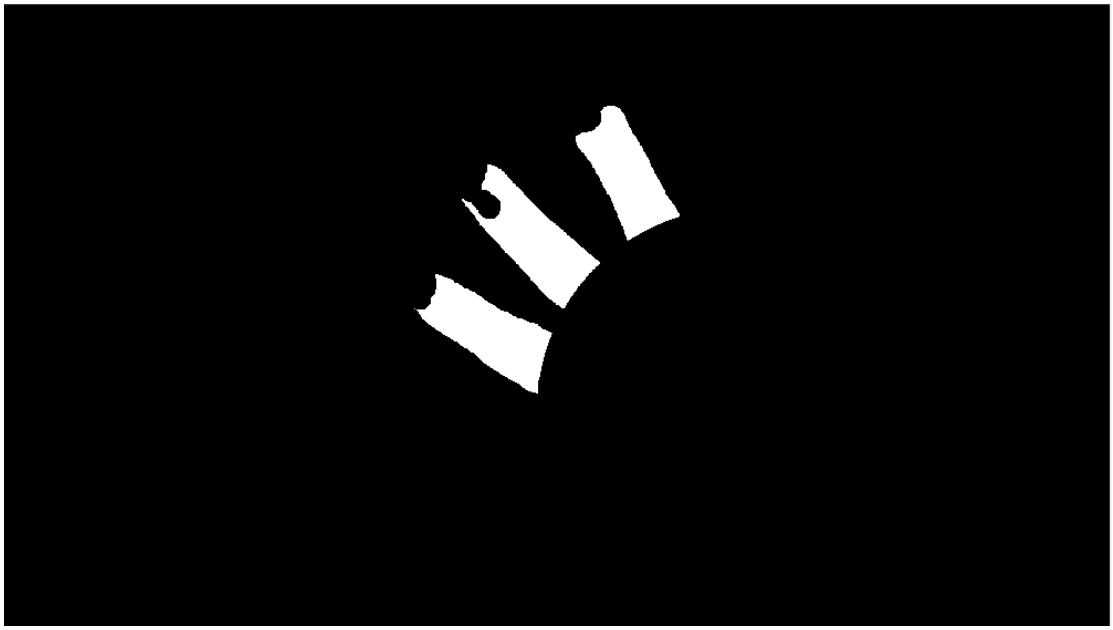


Figure 2.9 – final mask of 3 fingers

```
n =

     3
```

Figure 2.10 – original image of 4

Figure 2.11 – final mask of 4 fingers

```
n =

     4
```

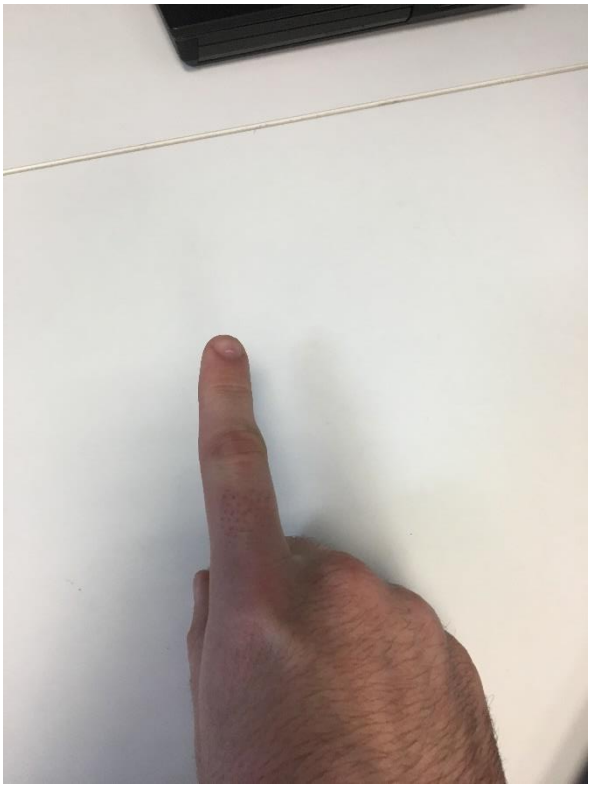


Figure 2.12 – original image of 1

אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

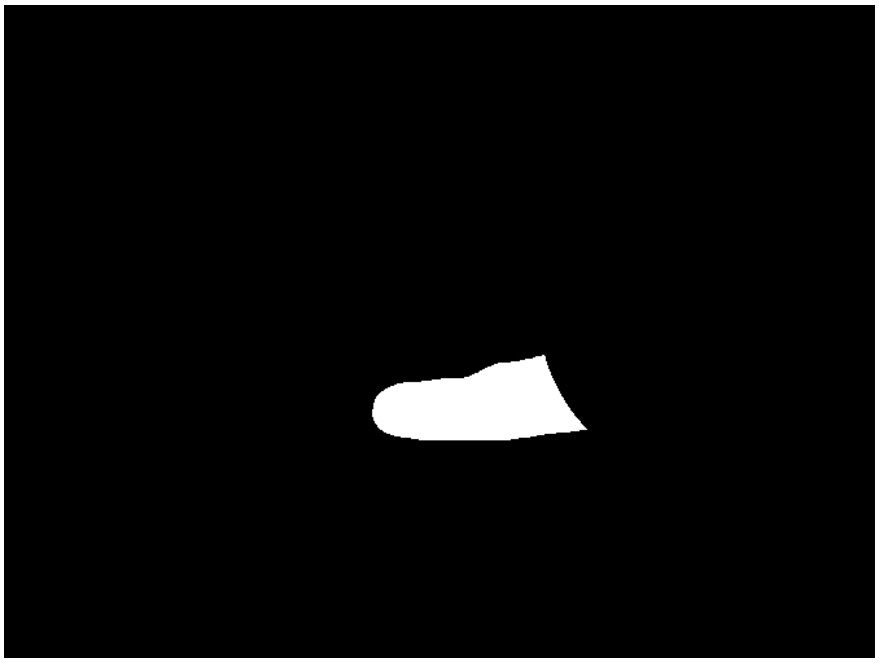


Figure 2.13 – final mask of 1 finger

```
n =

     1
```

Example of bad lighting & blurry image

Those images have a clear light environment with minimal shadowing, and the result of the algorithm is accurate is expected. Let's see how severe shadowing & blur (which we need to consider when dealing with a live feed of images taken of a moving object) :

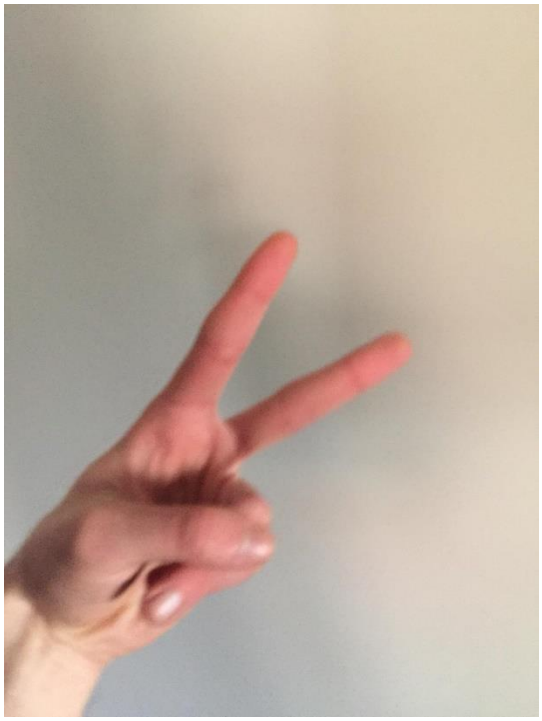


Figure 2.14 – Hard image of 2 fingers

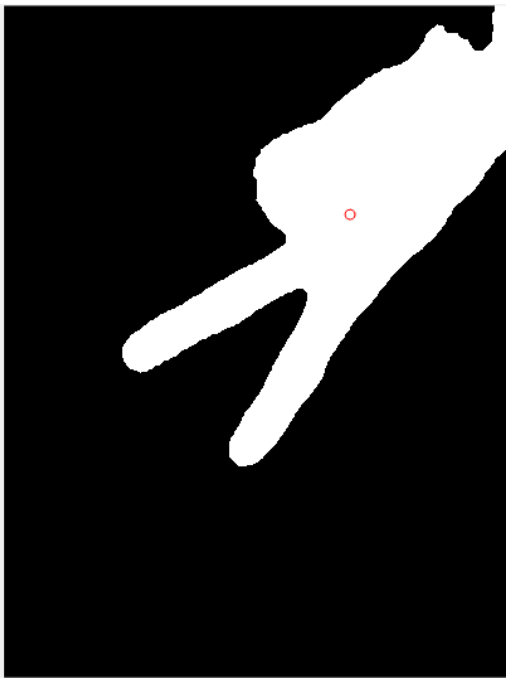


Figure 2.15 – initial binary mask

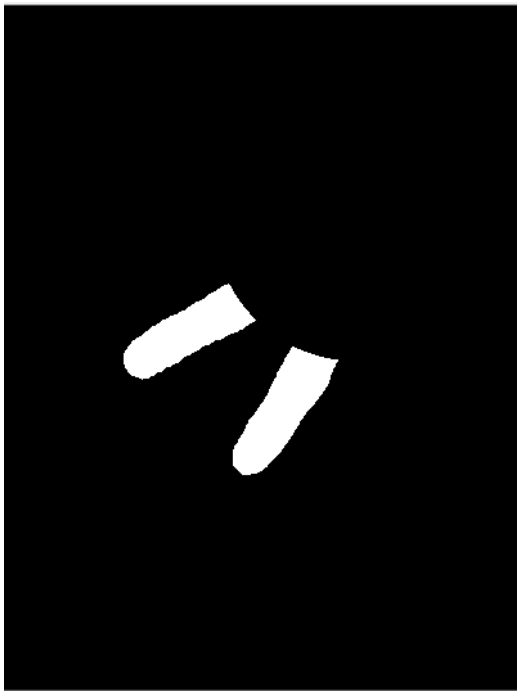


Figure 2.16 – final mask

```
n =

     2
```

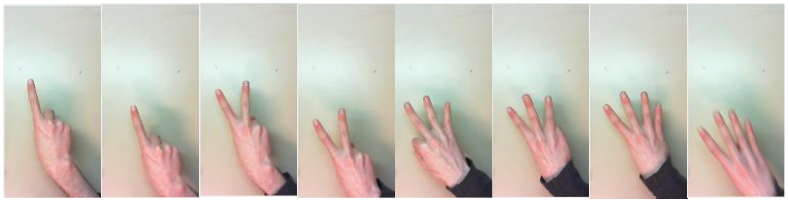
Run time & Accuracy

To get an approximation of the time that the algorithm takes to process the hand gesture & accuracy we loop through the detecting operation 10 times for each number, and insert a hand:

```
AccurecySum =

     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

= ~100% accuracy

Without figures the average time to detect the gesture is ~0.3s, and with display figures for debug ~4s.



Chessboard & new moves recognition

Background

After we got the skill level of the chess engine from the Hand Recognition part, we want to detect the initial position of the chess board. To do so, we need to find the chess board in the live feed scene of the overhead camera and extract each square in constant order to be sent for classification.

This part handles the recognition of the chess board in the frame, the extraction of each square to be sent to the piece classification (including an empty square) and detection of a new move that been made (for e.g. a hand in the frame shouldn't count as a new move nor define a new board state).

After calibration and geometric translation is done, we are using an internal MATLAB function called *detectCheckerboardPoints* [11] which detects a black and white checkerboard in a 2-D grayscale image. The function returns the detected points and dimensions of the checkerboard.

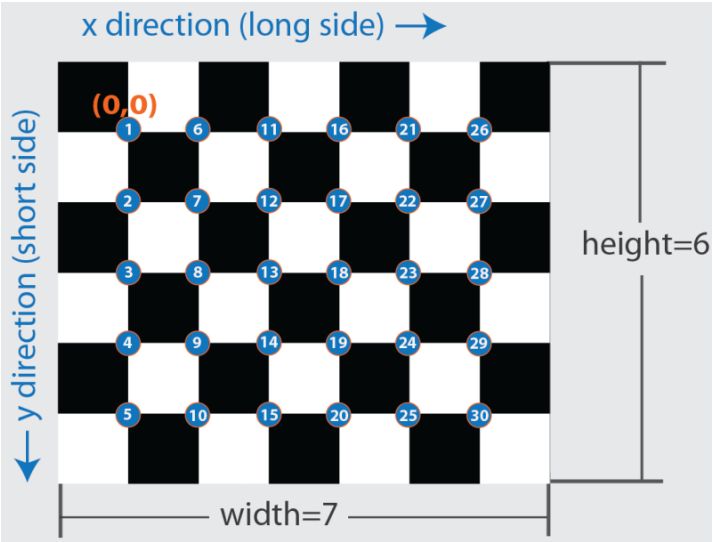


Figure 3.1 – e.g. output of the function detectCheckerboardPoints

This tool-kit which include this function is based on a fascinating work done on the subject of *'Automatic Camera and Range Sensor Calibration using a Single Shot'* by researchers in *Karlsruhe Institute of Technology* [12].

In general, throughout this section we are represent the chess board as 8x8 matrix with the pieces refers as follow (an example of the default initial position):

```
board =

  8×8 string array

    "r"    "n"    "b"    "q"    "k"    "b"    "n"    "r"
    "p"    "p"    "p"    "p"    "p"    "p"    "p"    "p"
    "*"    "*"    "*"    "*"    "*"    "*"    "*"    "*"
    "*"    "*"    "*"    "*"    "*"    "*"    "*"    "*"
    "*"    "*"    "*"    "*"    "*"    "*"    "*"    "*"
    "*"    "*"    "*"    "*"    "*"    "*"    "*"    "*"
    "P"    "P"    "P"    "P"    "P"    "P"    "P"    "P"
    "R"    "N"    "B"    "Q"    "K"    "B"    "N"    "R"
```

Which is identical to the FEN representation (page 51). Upper-case letters are the white pieces, and lower-case letters is the black pieces.

r	rook
n	knight
b	bishop
k	king
q	queen
p	pawn

However, with most of the testing regarding the pieces classification we will refer to the white pieces as the exact opposite (lower-case letters) and the black pieces with duplicate letters (due to labeling restrictions):

```
board =

  8×8 string array

    "rr"    "nn"    "bb"    "qq"    "kk"    "bb"    "nn"    "rr"
    "pp"    "pp"    "pp"    "pp"    "pp"    "pp"    "pp"    "pp"
    "*"     "*"     "*"     "*"     "*"     "*"     "*"     "*"
    "*"     "*"     "*"     "*"     "*"     "*"     "*"     "*"
    "*"     "*"     "*"     "*"     "*"     "*"     "*"     "*"
    "*"     "*"     "*"     "*"     "*"     "*"     "*"     "*"
    "p"     "p"     "p"     "p"     "p"     "p"     "p"     "p"
    "r"     "n"     "b"     "q"     "k"     "b"     "n"     "r"
```

In addition, the simplest way of tracking each side pieces are a simple matrix we are using for filtering noises of the other side pieces in each turn:

```
board =

  8×8 string array

    "2"    "2"    "2"    "2"    "2"    "2"    "2"    "2"
    "2"    "2"    "2"    "2"    "2"    "2"    "2"    "2"
    "*"    "*"    "*"    "*"    "*"    "*"    "*"    "*"
    "*"    "*"    "*"    "*"    "*"    "*"    "*"    "*"
    "*"    "*"    "*"    "*"    "*"    "*"    "*"    "*"
    "*"    "*"    "*"    "*"    "*"    "*"    "*"    "*"
    "1"    "1"    "1"    "1"    "1"    "1"    "1"    "1"
    "1"    "1"    "1"    "1"    "1"    "1"    "1"    "1"
```

We will encounter these representations on the subsequent pages in different situations.

In the following section, please bear in mind the next consideration:

Our accuracy of the piece's classification can be challenged (see page 44 for full analysis). Hence, we decided to disable the feature of starting for any starting position in the MATLAB side.

Its means that the line that gets us the initial matrix representation detected will currently give us the default starting position in chess instead of calling another function that classify each of the 64 squares on the board.

This doesn't mean we don't support this feature. Great efforts have been put in creating the logic Python side (checking for the legality of the position, communication, rendering, logic) and in MATLAB side (creating the FEN string represent a board state [page 51] , conversion of matrix representations and much more).

If the we want to consider an error margin of ~1 square in the initial position, we can support it. For now, and for simplicity sake of explaining the algorithm flow, think of the initial position as constant.

Algorithm flow

Detecting the Chessboard

This part is starting with the custom function `detectChessBoard` - A function that will call eventually to the built-in `detectCheckerboardPoints` function in a loop until a chess board (7x7 inner intersections = 8x8 squares) will be detected in the frame.

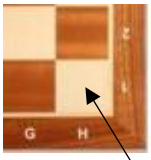
The first task is to identify a chess board in the frame. After that, we will use a geometric transformation functions to arrange the frame as we need (we want the white side to be in the bottom of the image).

Figure 3.2 – example of a frame from the camera

In `GetOnlyBoard` we are using `tform = fitgeotrans(movingPoints,fixedPoints,projective)` [that takes the pairs of control points, movingPoints and fixedPoints, and uses them to infer the geometric transformation specified by projective – when the fixed points is determine by a final constant size and the moving points is the intersection points detected] we can get an image with only the chess board.

Figure 3.3 – the chess board after geometric transformation & detected intersections

Then, we are using a custom function to get for each square the mean pixels value `img2meansquares` and can determine how to rotate the chess board frames as needed, even if there is a piece on the square.



White rook

It's worth noting that this approach can be **easily** (with the dynamic code and functions we wrote) replaced by classification of each of the 4 edge squares and determine where is the black & white rooks – and then rotate accordingly. Because we gained a high classification accuracy only towards the end of the development, other matters required our attention and we decided to leave this approach (which never fail us) as is.

After calibration and geometric translation is done we are returning the detected points and dimensions of the checkerboard accompanied with an appropriate voice instruction to arrange the starting position of the board. As mention, we decided to disable the feature of starting from any legal position (see page 47 for analysis).

Detecting a new move

We start from the custom function `GetNextFrame2` which containing 2 parts – ignoring false alarms like lighting changes, setup shaking, etc' and detecting when a move has been completed (for example ignoring insertion of a hand to the frame but not moving a piece etc.).

For all the following sections, we are looking only on the board area of the frame (using our custom function `GetOnlyBoard`).

First, we want to filter noises from lighting condition, slight shaking of the camera (e.g. someone knocking on the table, etc.). This is done simply by looping and checking 5 frame per seconds if the current frame is different from the initial frame in grayscale. If the change is below a certain threshold, then the change is not due to moving hands in the frame or moving pieces.

Initially the threshold was decided by try-and-error & looking at the difference matrix (of all the squares), later we simply took for a short period of time data from stationary position and checked for the max diff value represent a flickering of light, shadow, shaking table etc.

```
change_mat_th2: 8x8 double =

    5.7997    3.6220    6.3353    3.9835    6.3591    3.5101    6.5597    3.6648
    2.8596    2.5080    2.8299    2.3379    2.9648    2.1445    2.8505    2.3841
    1.9220    1.0154    1.7964    0.9750    1.7811    1.0523    1.9223    1.0009
    0.9672    0.8578    0.9218    0.9270    0.9890    0.9073    1.0783    0.8351
    1.8525    0.9834    1.8519    0.9150    1.6680    0.9456    1.9140    0.9415
    1.1137    0.8368    0.9718    0.8458    0.8932    0.7843    0.8804    0.8714
    5.1105    2.4569    3.7037    2.0782    4.3039    2.0972    4.3740    2.3469
    2.8910    2.7575    2.4548    2.9671    2.7813    2.7321    2.8261    2.5866
```

Figure 3.4 – example of the difference matrix of noise – just waving under the camera

Secondly, if a 'real change' has been detected (e.g. a hand inserted to the frame) we are waiting for a stationary position to occur. Similarly, this is achieved using a threshold over the sub of the current frame and the previous frame, over the course of *num/30* pause (passing num=60 to the function). we are giving the user ~2s to not have a hand in the frame that isn't moving (e.g. touching a piece without moving your hand for too long).

A regular scenario is inserting a hand to move a piece and move the hand away – throughout those scenarios there is not sequential frames(2s apart) that have a max diff less than some threshold.

Hence, in average we are waiting approximately 2s after each move to have a stationary position safely (consider the time of the reach back your hand from the piece).

```
change_mat_64: 8x8 double =

    6.0810    2.9479    5.7147    3.1129    5.2673    2.5661    5.3348    2.7119
    2.5106    2.3771    2.7209    2.2481    2.5699    2.1941    2.3264    2.2268
    2.7372    1.4919    2.7417    1.4730    2.6707    1.3163    2.8605    1.3800
    1.4103    1.2832    1.4960    1.6273    1.8132    1.2928    1.6646    1.4257
    2.6766    1.3025    3.0426    1.7332   62.3442    3.2133    3.4307    1.6785
    1.3792    1.2154    1.4072    1.5611    3.6383    1.5535    1.5100    1.5805
    4.5190    1.9529    3.8158    2.0237   55.7365    1.7169    4.9418    2.0012
    2.5164    2.8740    2.0154    2.5570    8.9261    2.7034    2.3831    2.5493
```

Figure 3.5 – example of the difference matrix of a move – e2e4

Next, the stationary frame returned from `GetNextFrame2` is been proceed to the function `GetNewPositions2`.
This function responsible for giving information about the new move detected - the 'from' square and the 'to' square.

Also, this function needs to handle castling, all this with a several logic steps of image processing and classification.

In general, the idea is that if the regular thresholds have a hard time to determine which square was the piece moving from (for e.g.), then we will continue to classification filtering process.

Detecting the move 'from' square

`GetNewPositions2` square is getting the new image of the board and the old one from the previous function `GetNextFrame2.`
First, we will compute the absolute subtraction of the grayscale frames (we don't need the details RGB provides, and it's easier to work on grayscale color space when tracking changes in our case).

Tests shows that because our pieces having a white background that is similar to the chess board white squares, a bump to the diff values in the white squares is made (multiply by 2).

Now we can start the process of figuring out what square the piece has moved from.

The first step is to mask-on the diff-matrix with the pieces of the side to move, which means filtering every square in the matrix that has an opposite color piece (we are tracking that). Its means ending the choice of the source square to be an opposite color piece. This step is logical at first, but after gaining the capability of classify pieces at high accuracy, we can change this step to simply judge between candidates by checking even the opposite pieces in tradeoff with speed, simplicity, and error-margin.

If we had a 100% accuracy classification, many logical layers could be replaced from this step and the steps to follow. Either way, we present here an algorithm that works, and accept other ways that would work with some tradeoff.

```
change_mat_64_turn: 8x8 double =

         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0
    4.7493    2.3760    3.2980    1.5866   80.1634    1.6293    3.8799    1.9595
    2.6296    2.6401    2.0633    2.3236    2.4077    2.3129    2.1866    2.2370
```

Figure 3.6 – diff-matrix after masking, white to move, 'from' detection step 1

Next, we are extracting the diff-values that are at-least 60% from the max value and their index squares (throughout the code we gave the square an index from 1 to 64).

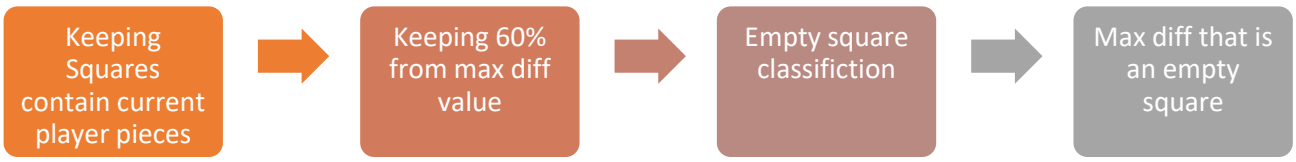
```
board =

     1     9    17    25    33    41    49    57
     2    10    18    26    34    42    50    58
     3    11    19    27    35    43    51    59
     4    12    20    28    36    44    52    60
     5    13    21    29    37    45    53    61
     6    14    22    30    38    46    54    62
     7    15    23    31    39    47    55    63
     8    16    24    32    40    48    56    64
```

We chose to check all those top squares because not always the max diff value was the right square source, and the actual difference when a black/white piece move from white/black square can differ.

If there are more than 1 suspect squares, for each square we are looking for an empty square using classification on the dataset called *'categoryClassifier5'* (see page 39), which present us suffice result on finding if a square is empty or not.

If there are more than 1 square that are labeled as empty in the new frame (after classification), we are choosing the one that have the greater diff (from the precious state) of them.

All those layers of filtering making it very likely to succeed, considering the fact that the source square (the 'from' square) will be classify as empty in the new frame, will have a big difference comparing to the old state of the board and will have the current side-to-move piece on it in the previous step (the first masking).



אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

Detecting the move 'to' square

Logically, the detecting of the 'to' square is similar to the detection of the 'from' square. The first step is to mask-off the diff-matrix with the pieces of the side to move, which means filtering every square in the matrix that has the current player color piece.

Its means ending the choice of the destination square to be already with a current-side color piece. This step is logical at first, but after gaining the capability of classify pieces at high accuracy, we can change this step to simply judge between candidates by searching the piece that was on the 'from' square in the earlier state of the board - even the opposite pieces in tradeoff with speed, simplicity, and error-margin.

```
change_mat_64_turn_to: 8x8 double =

    5.8436    3.1693    5.6805    3.3248    5.9168   49.1241    5.9000    3.1424
    2.6569    2.4608    2.4836    2.0678    0.9854   34.6697    2.5028    2.1579
    1.8162    0.9529    1.8814    0.9471    1.9043    1.0059    1.9817    0.9555
    1.0544    0.8773    1.0178    0.8493    2.0672    0.8732    0.9997    0.9367
    1.8994    1.0687    1.7311    0.9816         0    0.9968    1.5818    1.0279
    1.0469    0.8927    0.9235    0.9166    0.8712   39.2285    0.9325    0.9331
         0         0         0         0    1.6430         0         0         0
         0         0         0         0         0         0         0         0
```

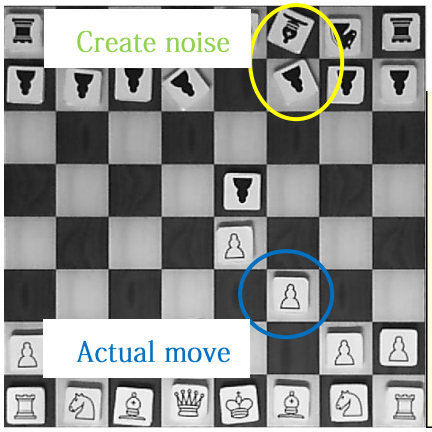
Figure 3.7 – diff-matrix after masking, white to move, we spin a little bit some black pieces to create **uncertainty** which resolve with classification

Next, we are extracting the diff-values that are at-least 60% from the max value and their index squares (throughout the code we gave the square an index from 1 to 64).

We chose to check all those top squares because not always the max diff value was the right square source, and the actual difference when a black/white piece move from white/black square can differ.

If there are more than 1 suspect squares, for each square we are getting the prediction score of matching the piece on the 'from' square in the earlier state, using classification on the dataset called *'categoryClassifier7'* (see page 37), which present us suffice result on classify the pieces from one another.

Then, we are choosing the square with the highest prediction score to match.

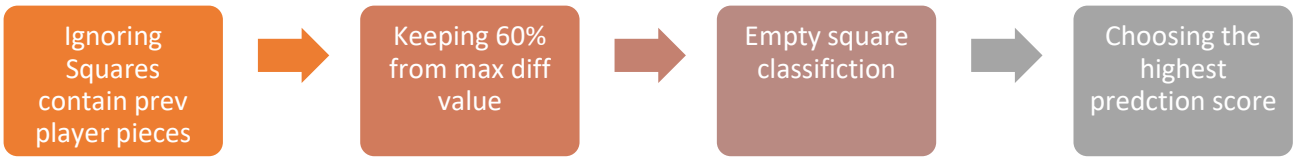



```
change_mat_64_turn_to_maybe_max: 8×8 logical array =

   0   0   0   0   0   1   0   0
   0   0   0   0   0   1   0   0
   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0
   0   0   0   0   0   1   0   0
   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0
```

```
empty_maybe_to: 1x3 double =

   -0.1467   -0.1608   -0.0617
```

Figure 3.8 – prediction scores, when the second index is the actual piece that moved (white pawn)

All those layers of filtering making it very likely to succeed, considering the fact that the destination square (the 'to' square) will be likely to classify as the same piece that moved in the new frame, will have a big difference comparing to the old state of the board and will have the current side-to-move piece on it in the current frame.



In this context, it's worth noting that if we detect 2 special squares that belongs to the castling rights (king & rooks moved = 4 squares changed), we are feeding the engine with the move that tied with the king skipping 2 squares. This logic also can have classification for more safety and accuracy.

Illegal move loop

After detecting the user move, we are feeding it to the chess engine via the text-based communication with the python side (page 50). Then, a blocking section are waiting for the engine respond of the best move. If the current move was illegal, logic on the python side will pick it up and send through the communication pipe 'illegal move' instead of a regular log of suggestion moves.

Once the user is getting the UI feedback of an illegal move (accompanied by proper sound) the MATLAB set a flag that enable a state we called 'illegal move loop'.

In this state, each time the mistaken player makes a new move, we figure out the new position and comparing it to the last legal position (in MATLAB).

Until the last legal position will be restoring on the board (using the full matrix representation of pieces), the user will keep hearing 'illegal move' sound effect.

Once the user will restore to the legal position (which also present on the UI web page) a proper sound will be produce, which gives the user a sign that the system is ready for him to make a new move.

Live results

Video attached in the Result section, see page 59.

אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

Board pieces classification

Background

In our project, classification of pieces has a big role helping us judge in uncertainty situation, or fully recognize which piece and where are present on the physical chess-board.

A goal of this project is responsive identification of both the board and pieces using image processing. This research's [1] [2] and many more provide some applicable approaches to identifying 3D chess pieces using Convolutional Neural Network (CNN) - due to insufficient results from more robust image-processing algorithm (e.g. SURF/SIFT).

A Convolutional Neural Network (CNN) is a powerful machine learning technique from the field of deep learning. CNNs are trained using large collections of diverse images. From these large collections, CNNs can learn rich feature representations for a wide range of images. These feature representations often outperform hand-crafted features such as HOG, LBP, or SURF [15].

For the goal of using those image-processing tools, the using of 3D pieces was ruled out. We decided to approach the piece identification part of the problem by using 2D pieces (which is widely used for learning chess) & features from the SURF as taught in the classroom and assignments,

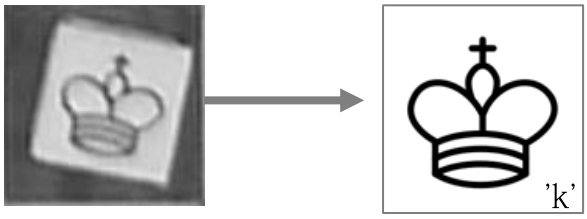
In computer vision, speeded up robust features (SURF) is a patented local feature detector and descriptor. It can be used for tasks such as object recognition, image registration, classification, or 3D reconstruction [13].

To detect interest points, SURF uses an integer approximation of the determinant of Hessian blob detector, which can be computed with 3 integer operations using a precomputed integral image. Its feature descriptor is based on the sum of the *'Haar wavelet'* response around the point of interest. These can also be computed with the aid of the integral image.

אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

SURF descriptors have been used to find and recognize objects, people or faces, to reconstruct 3D scenes, to track objects and to extract points of interest.

Bag of words or Bag of features is a technique adapted to computer vision from the world of natural language processing. Since images do not actually contain discrete words, we first construct a "vocabulary" of SURF features representative of each image category [14]. Visual image categorization is a process of assigning a category label to an image under test. Categories may contain images representing just about anything, for example, dogs, cats, trains, boats. In our case, we will train the classifier for the different types of chess-pieces.



Analysis

Datasets

The datasets have been created using the *GetOnlyBoard* function, and extracting each square exactly with the same quality, same height & range of 2~3 exposure levels (to emulate different lighting conditions).

Due to results of the training and feature extracting algorithms we are using 2 different datasets (trained differently) – one for piece classification (including differentiate colors) and the second for determine if a square is empty or not. We will show the analysis that make us choose that way.

The piece classifier containing images of the pieces on squares (same scale, just cover the board with paper sheets) with white background – to maximize features of the piece itself and minimize features of the background square color detected (a piece is likely to spin in the square, which can affect accuracy).

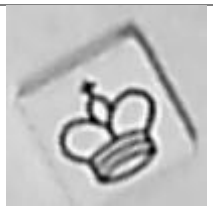
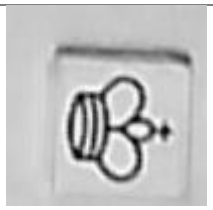
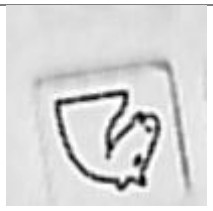
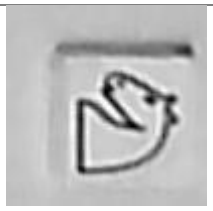
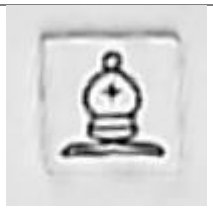
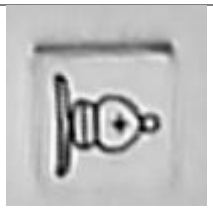
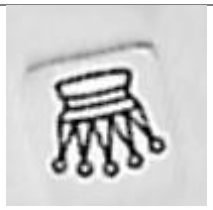
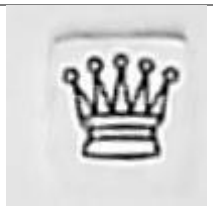
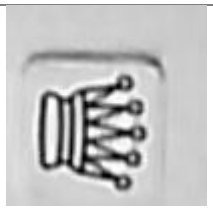
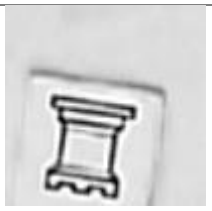
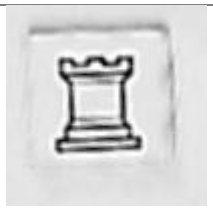
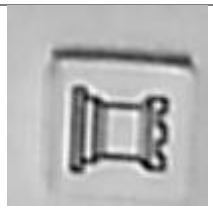
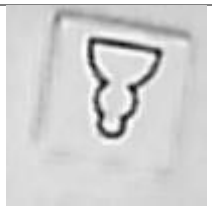
We also make sure to not make a huge dataset to avoid overfitting (too much features) and to make a varied choice of angles of the pieces.

categoryClassifer7

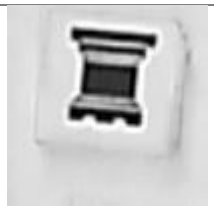
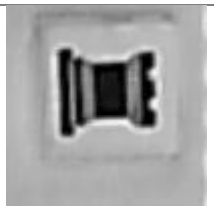
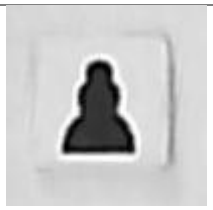
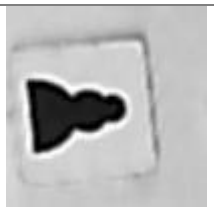
Used to classify pieces, consist of total 230 images

Label	Count
b	16
bb	18
k	9
kk	9
n	16
nn	18
p	36
pp	56
q	9
qq	9
r	16
rr	18

אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

Piece name	Label	e.g. 1	e.g. 2	e.g. 3
White king	'k'			
White knight	'n'			
White bishop	'b'			
White queen	'q'			
White rook	'r'			
White pawn	'p'			
Black king	'kk'			

אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

Black knight	'nn'			
Black bishop	'bb'			
Black queen	'qq'			
Black rook	'rr'			
Black pawn	'pp'			

categoryClassifer5

Used to classify empty squares, consist of total 1075 images.

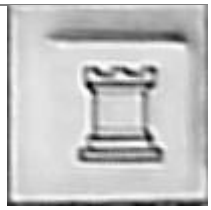
This dataset is trained using SURF features extracted over a regular grid of point locations at multiple scales (we are using different method from the piece classification that can't find features on empty squares, we will elaborate on this in the next section).

In addition, we want to make sure to have a regular squares from the live feed frames to be able to distinguish a piece from empty square.

Label	Count
b	66
bb	76
e	84
ee	91
k	38
kk	37
n	73
nn	78
p	150
pp	172
q	44
qq	22
r	71
rr	73

Piece name	Label	e.g. 1	e.g. 2	e.g. 3
White king	'k'			
White knight	'n'			

אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

White bishop	'b'			
White queen	'q'			
White rook	'r'			
White pawn	'p'			
Black king	'kk'			
Black knight	'nn'			
Black bishop	'bb'			

אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

Black queen	'qq'			
Black rook	'rr'			
Black pawn	'pp'			
White square	'e'			
Black square	'ee'			

אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

Creating a classifier

Our script to create a classifier object called 'creatclass' which following the steps form the MATLAB guide to create an Image Category Classification Using Bag of Features [14].

1. Construct an ImageDatastore based on the categories `{'b','bb','nn','n','p','pp','q','ee','e','qq','r','rr','k','kk'}`. ImageDatastore helps us manage the data.

2. Optionally: Separate the set into training and validation data using `[trainingSet, validationSet] = splitEachLabel(imds, 0.85, 'randomize');`
   Separate the sets into training and validation data. Pick 85% of images from each set for the training data and the remainder 15% for the validation data. Randomize the split to avoid biasing the results.

3. Create a Visual Vocabulary and Train an Image Category Classifier. Bag of words is a technique adapted to computer vision from the world of natural language processing. Since images do not actually contain discrete words, we first construct a "vocabulary" of SURF features representative of each image category.

   This is accomplished with a single call to *bagOfFeatures* function, which:

   3.1 extracts SURF features from all images in all image categories

   3.2 constructs the visual vocabulary by reducing the number of features through quantization of feature space using K-means clustering.

   Note: the method used to define point locations for feature extraction for the classfierCategory7 is using the argument *'PointSelection' = 'Detector'*. When set to 'Detector', the feature points are picked using SURF feature detector. Otherwise, the points are picked on a regular

אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

grid with spacing defined by *'GridStep'* (default [8 8]). This method gave us better accuracy, as shown later.

4. Encoded training images from each category are fed into a classifier training process invoked by the `trainImageCategoryClassifier` function. This function relies on the multiclass linear SVM classifier from the Statistics and Machine Learning Toolbox™.

   An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall [17].

   The above function utilizes the encode method of the input bag object to formulate feature vectors representing each image category from the *trainingSet*.

Classify pieces

Key points on creation of this classifier:

```
Extracting features from 230 images...done. Extracted 5350 features.

Keeping 80 percent of the strongest features from each category.

Balancing the number of features across all image categories to improve clustering.
* Image category 4 has the least number of strongest features: 150.
* Using the strongest 150 features from each of the other image categories.

Using K-Means clustering to create a 500 word visual vocabulary.
Number of features          : 1800
Number of clusters (K)      : 500
```

Evaluating image category classifier for 12 categories (100% training, validation set is 868 images from categoryClassifier7 dataset):

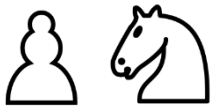
	PREDICTED											
KNOWN	b	bb	k	kk	n	nn	p	pp	q	qq	r	rr
b	0.94	0.02	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00
bb	0.00	0.92	0.00	0.01	0.00	0.00	0.00	0.01	0.00	0.00	0.03	0.03
k	0.00	0.00	0.94	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.03	0.00
kk	0.00	0.00	0.03	0.94	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03
n	0.00	0.00	0.00	0.00	0.99	0.00	0.01	0.00	0.00	0.00	0.00	0.00
nn	0.00	0.00	0.00	0.00	0.00	0.94	0.00	0.03	0.00	0.00	0.00	0.04
p	0.00	0.00	0.00	0.00	0.13	0.00	0.77	0.05	0.00	0.02	0.04	0.00
pp	0.01	0.00	0.00	0.01	0.01	0.01	0.00	0.94	0.00	0.00	0.00	0.04
q	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.93	0.07	0.00	0.00
qq	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00
r	0.00	0.00	0.02	0.00	0.00	0.02	0.00	0.02	0.00	0.00	0.94	0.02
rr	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.06	0.00	0.00	0.00	0.94

```
* Average Accuracy is 0.93.
```

The overall accuracy is decent, though 93% accuracy means usually 1-2 pieces margin error on new positions.

This is actually an impressive result, considering that the validation set has been taken from 3 different rooms with different light condition & 2-3 different exposure settings (without overlapping images from the training set).

We highlight the inconsistent result of the white knight and the white pawn, probably due to their design similarity and the lack of details inside (they share some curves on the edges) considering the final square images is in low-resolution and smudgy.



We also play with the training process to see what work best. An idea was to combine the pieces without separating the colors (we can still use those type of classification) – the result wasn't better (86% accuracy). When we used the *'GridStep'* option we could choose 8x8 kernels or other sizes, changing it wasn't effective (bear in mind that the actual square is blurry after the digital zoom, so we don't want to extract features from too close or too far).

Classify empty squares

Key points on creation of this classifier:

```
Extracting features from 1075 images...done. Extracted 734400 features.

Keeping 80 percent of the strongest features from each category.

Balancing the number of features across all image categories to improve clustering.
* Image category 12 has the least number of strongest features: 12227.
* Using the strongest 12227 features from each of the other image categories.

Using K-Means clustering to create a 500 word visual vocabulary.
Number of features          : 171178
Number of clusters (K)      : 500
```

Evaluating image category classifier for 14 categories (85% training 15% validation):

	PREDICTED													
KNOWN	b	bb	e	ee	k	kk	n	nn	p	pp	q	qq	r	rr
b	0.90	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.10	0.00	0.00	0.00	0.00	0.00
bb	0.00	0.91	0.00	0.00	0.00	0.00	0.09	0.00	0.00	0.00	0.00	0.00	0.00	0.00
e	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
ee	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
k	0.00	0.00	0.00	0.00	0.50	0.17	0.33	0.00	0.00	0.00	0.00	0.00	0.00	0.00
kk	0.00	0.00	0.00	0.00	0.00	0.50	0.00	0.50	0.00	0.00	0.00	0.00	0.00	0.00
n	0.00	0.00	0.00	0.00	0.00	0.00	0.91	0.00	0.00	0.00	0.09	0.00	0.00	0.00
nn	0.00	0.08	0.00	0.00	0.00	0.00	0.00	0.83	0.00	0.08	0.00	0.00	0.00	0.00
p	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.91	0.09	0.00	0.00	0.00	0.00
pp	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.04	0.00	0.96	0.00	0.00	0.00	0.00
q	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.71	0.29	0.00	0.00
qq	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00
r	0.00	0.27	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.73	0.00
rr	0.00	0.09	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.91

```
* Average Accuracy is 0.84.
```

As we can see, the overall accuracy isn't great if we are looking from this classifier to tell us which piece is on a particular square, but – we can see that we gained 100% accuracy telling us if a square is empty or not (used in page 29).

Classify the starting position

Full examples of the results on some boards:

Figure 4.1 – original snapshot of the board e.g.

, rr, rr, rr	, nn, nn, nn	, bb, bb, bb	, kk, kk, kk	, qq, qq, qq	, bb, bb, bb	, nn, nn, nn	, rr, rr, rr
, pp, pp, pp	, pp, pp, pp	, pp, pp, pp	, pp, pp, pp	, pp, pp, pp	, pp, pp, pp	, pp, pp, pp	, pp, pp, pp
, pp, pp, pp	, pp, pp, pp	, pp,pp,pp	, pp, pp, pp	, pp, pp, pp	, pp, pp, pp	, pp, pp, pp	, pp, pp, pp
*	*	*	*	*	*	*	*
*	*	*	*	*	*	*	*
, p, p, p	, p, p, p	, p, p, p	, p, p, p	, p, p, p	, nn, nn, nn	, p, p, p	, p, p, p
, p, p, p	, p, p, p	, p, p, p	, p, p, p	, p, p, p	, p, p, p	, p, p, p	, p, p, p
, r, r, r	, n, n, n	, b, b, b	, k, k, k	, q, q, q	, p, p, p	, n, n, n	, r, r, r

Figure 4.2 – pieces classification e.g.

We can see that we have 2 errors in the position classification, especially when we had some distortion on the board (square f1 on the white bishop).

אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

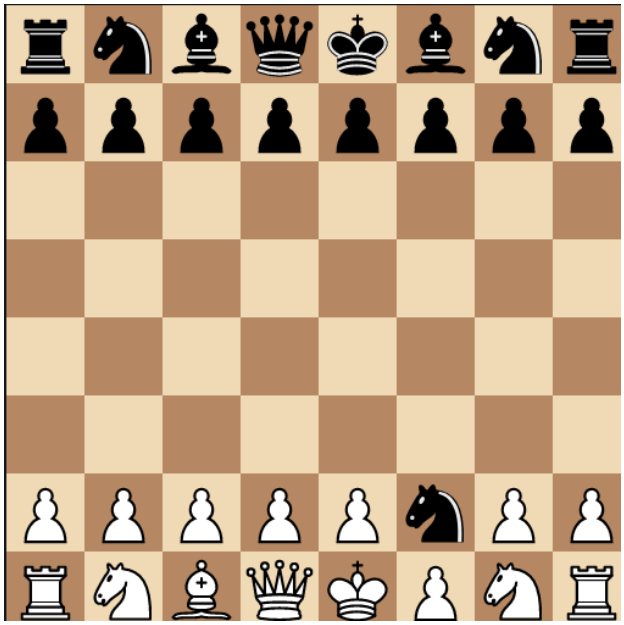


Figure 4.3 – error margin of 2 pieces, e.g

Our accuracy of the piece's classification can be challenged. Hence, as mention before, we decided to disable the feature of starting for any starting position in the MATLAB side.

Microframework & Chess engine communication

Background

To connect an analysis of a chess engine, we need to be able to use an UCI communication protocol. In addition, we want to be able to maintain a User Interface that will be interactive and simple for providing feedback on the concurrent game played.

Our approach is to maintain a microframework (simple web app) locally, communicate with the image-processing part (MATLAB) with text-file read\write based communication & tracking the chess game in Python with the help of the chess-python library. We made changes to the free open source library to fill our need best, which we will elaborate later.

The python code handles communicating with the chess engine, maintaining a web app with chessboard rendering, best-move suggestion and legality feedback.

Definitions

Asynchronous I/O

Data transfers can be synchronous or asynchronous. The determining factor is whether the entry point that schedules the transfer returns immediately or waits until the I/O has been completed.

In our case, there are many sections in the python code that waiting for the MATLAB code to write to several files, for e.g. write the detected move by the player. Upon those bloc king points, we are entering a state of 'busy wait' with short sleep duration.

אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

Python-chess library

python-chess [5] is a pure Python chess library with move generation, move validation and support for common formats.

This library helps us to track the current game state, attach a local chess engine to analyze the game, check for legality of the moves before feeding the engine the new state of the game, rendering an SVG object that describe the board state (which will be sent to the web page) etc. Later we will elaborate on the changes we made in this library to fit our needs and design.

Multithreading

Multithreading is a widespread programming and execution model that allows multiple threads to exist within the context of one process. These threads share the process's resources but can execute independently.

In our design, there are 2 threads in the python code – 1 that handles the server requests and updates via socketIO and 1 that handles the text-file based communication with the MATLAB code.

Text-based communication

A text-based protocol or plain text protocol is a communication protocol whose content representation is in human-readable format.

The idea is simply enabling the 2 processes running on the same pc (python code & MATLAB code) to communicate with each other. This is accomplished by reading and writing to the same files that represent the player move, the engine best-move suggestion, legality of the moves, FEN starting position etc.

If a certain stage requires the MATLAB code to wait for the response of the Python code, then we will block the MATLAB process until a new line will be written to the file. Such design needs to be tracked carefully to be bug-free.

אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

Microframework

A microframework is a term used to refer to minimalistic web application frameworks. It is contrasted with full-stack frameworks.

Typically, a microframework facilitates receiving an HTTP request, routing the HTTP request to the appropriate controller, dispatching the controller, and returning an HTTP response. Microframeworks are often specifically designed for building the APIs for another service or application.

To create our microframework, we are using Flask [6] - a micro web framework written in Python.

Flask enables us to maintain local server with html and CSS driven web page. Each process that will enter our local address will refer as a client (e.g. Firefox browser, Chrome browser etc.).

SocketIO

Flask-SocketIO [7] gives Flask applications access to low latency bi-directional communications between the clients and the server. The client-side application can use any of the SocketIO official client's libraries any compatible client to establish a permanent connection to the server.

In our case this service enables the chess-game tracking thread in python feeding the web page with updates about the current state of the game, legality feedback of the user detected move etc.

FEN string

Forsyth – Edwards Notation (FEN) is a standard notation for describing a board position of a chess game. The purpose of FEN is to provide all the necessary information to restart a game from any position. Here is the FEN for the starting position:

rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1

To enable a feature that detect any starting position of the board and continue playing from there, we code basic function (*board2FEN.m*) to convert 8x8 board structure (part of the MATLAB tracking the game) to FEN string in the beginning of the game, which then sent to the python side via text-file. The web page will give a feedback if the position is legal. If not, the python will enter a state that waits for the MATLAB to rewrite the file for a new FEN string.

CSS

CSS is a style sheet language used for describing the presentation of a document written in a markup language like HTML.

HTML

Hypertext Markup Language (HTML) is the standard markup language for creating web pages and web applications. With Cascading Style Sheets (CSS) and JavaScript, it forms a triad of cornerstone technologies for the World Wide Web.

jQuery

jQuery is a JavaScript library designed to simplify HTML DOM tree traversal and manipulation, as well as event handling, CSS animation and more.

jQuery enable us to write function for flask-soketIO to emit changes to the web page (*main.js*).

Stockfish

Stockfish is a free and open-source [8] UCI (Universal Chess Interface [9]) chess engine, available for various desktop and mobile platforms.

Stockfish is consistently ranked first or near the top of most chess-engine rating lists and is the strongest open-source chess engine in the world

Python-chess library help us to communicate with the chess engine, which runs locally on our PC.

In the beginning of the game, we are feeding the engine with the level detected in the hand-recognition part in MATLAB (page 9).

Algorithm flow

Starting the flask app server

The first section of the python code is starting a thread that handles a flask socketIO app. This app represents as a web page running locally (address http://127.0.0.1:5000/). The main route of the web page is loading the *index.html* file. For now, the app will listen to any client want to connect (in our case a simple web browser process like Firefox). Updating the web page will be through socketIO commands.

> *socketio.emit('enginelevel', {'level': skill_level}, namespace='/test')*
> sending to *'enginelevel'* function in main.js (jQuery) file a *skill_level* variable to the argument *'level'* of the function.

All the clients connected will be updating at the same time via post/get HTTP requests.

```
Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [16/Feb/2019 20:31:58] "GET / HTTP/1.1" 200 -
```

Initialize the board object

Once a client is connected via the address, a new thread is created of the type *'chessThread'* . We code this class to have a simple structure in the python code. The thread will start from the function *run()*.

Class chessThread

A class that represent a thread type and its functions.

Run()

initalize()

socketio.emit('newgame_sound', namespace='/test')

self.gameOn()

initialize()

Initializing local variables, attaching a local chess engine to the *board* object (default evaluation time *'evaltime'=0.5s*), waiting for the hand recognition part to determine the engine level (default: 15), then waiting for the MATLAB to recognize the starting position of the game (transferring a FEN string with the text-file communication, file name *'Fen.txt'*). If the position is illegal (checking is done using python-chess library), this part waiting in busy-wait-sleep loop for a new FEN to be transferred.

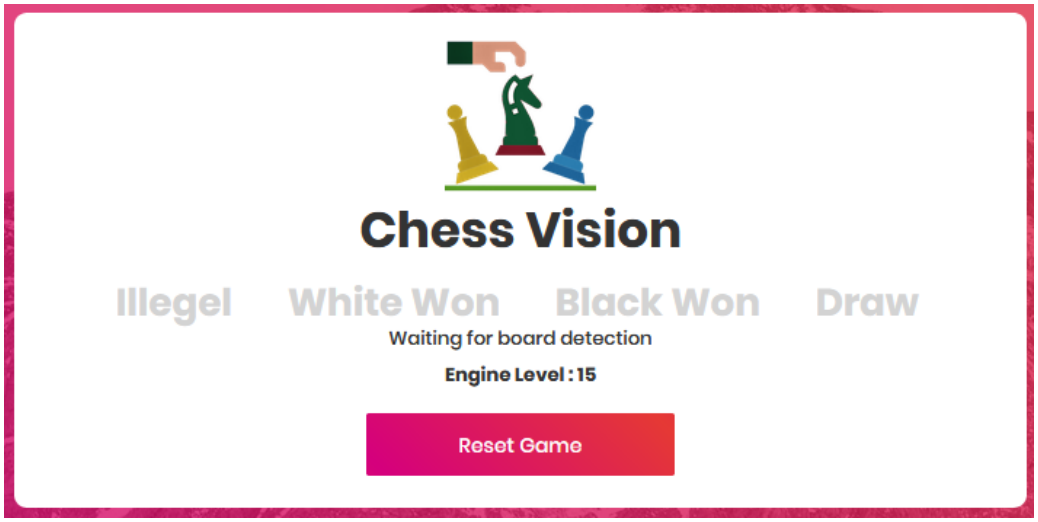


Figure 5.1 – starting state of the UI web page

Once a legal position has appeared on the physical board, we are updating the web page, initializing a *board* object, feeding the engine of the new game state and moving on to the *'gameOn()'* function.

While the game isn't over

Wait for a user move

Check the legality of the move

Feed the engine

Render the board

Update the web app

gameOn()

Busy-waiting (with some sleep delay) to a new move detected in the MATLAB section. If the move is legal, we push it to the current state of the board and rendering the board again with the new suggestion of the chess engine and emitting the change via *'newmove'* function in main.js.
Otherwise (illegal move) we dont push this move to the game. We update the UI to notify for illegal move accompanied with appropriate sounds.

Once the game is over

Emit the web page with the results and close the current chess thread. The user can start a new game by pressing the *'Reset'* button.

UI

Best move suggestions are illustrated with an arrow. The board rendering considering checks on the king and highlights of the from-and-to square of the last move.

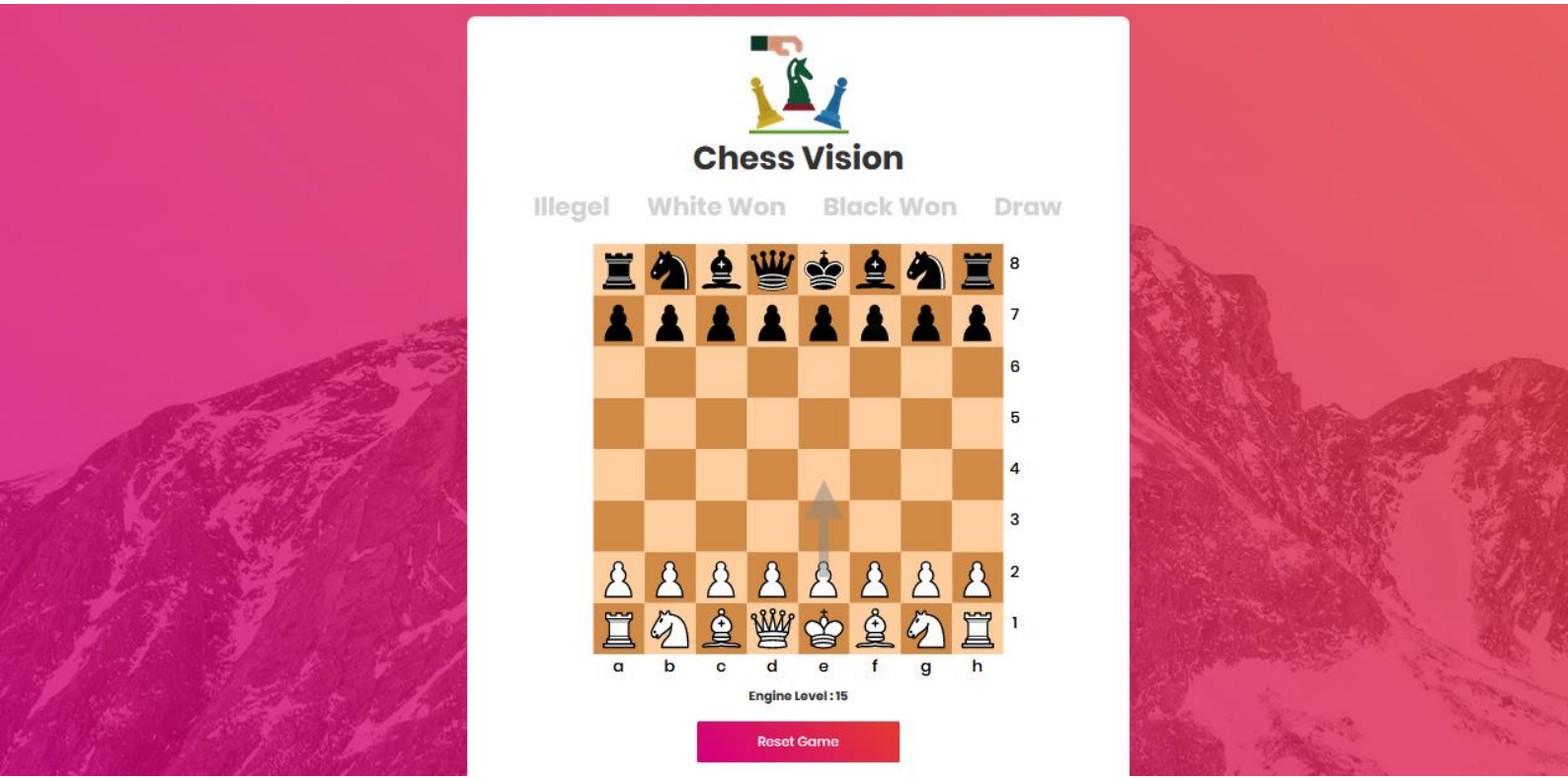
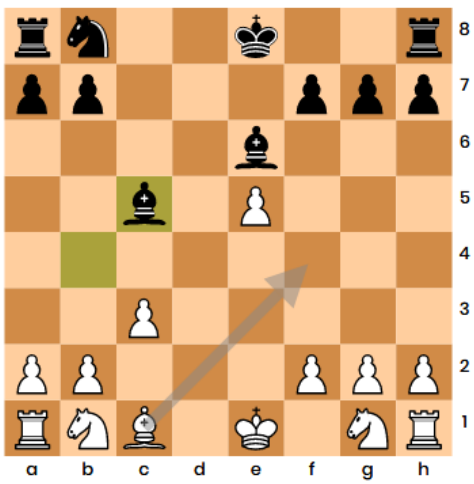
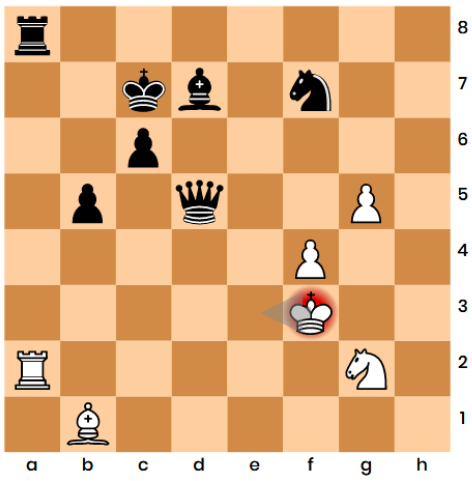


Figure 5.2 – final UI





Figures 5.3 - highlights of the from-and-to square highlight of check

Elaboration on some functions

Library: chess__init__.py

king_squaree_ifcheck()

```
def king_squaree_ifcheck(self):
    # Custom function - if the current side is in
                check return the king square
    check = self.king(self.turn) if self.is_check()
                    else None
    return check
```

- Custom function - if the current side is in check return the king square

Library: chess\svg.py

board()

- Changed rendering of the side notation in 'if coordinates' statement.

socketFlask,py – main file

```
def getMoveAttribute(move_str, square_number):
    """input: move_str = uci move e.g.
'e2e4',square_number : 0/1 for the first/second
square"""
    """output: chess.E2 / chess.E4 attribute for
drawing arrow on this move"""

    str = move_str[0 + square_number * 2:2 +
       square_number * 2]
    return getattr(chess, str.upper())
```

- Custom function to convert easy to read notation of moves to the square attributes of the class board.

```
def writeMove():
    """Write the current engine suggestion to the
EngineMoves txt file"""
    global currentBestMove
    m = open(file_to_write, 'a')
    m.write(currentBestMove + '\n')
    m.close()
```

- Keep a log of the suggestion moves by the engine. This can be used by any third-party part for future use.

main.js – jQuery main file

```
//receive board state from server
socket.on('newmove', function(msg) {
    console.log("Received board" + msg.board);
    $('#board').html(msg.board);
});
```

- With socketIO we can change the index.html file for all connected clients. In the function, we are emitting a new board rendering after a legal move.

Usage

Python 3.7.0 [10]. To run the python side, we need to install a few packages.

```
pip install flask-socketio
pip install flask
pip install threading
pip install python-chess
```

```
from flask_socketio import SocketIO, emit
from flask import Flask, render_template, url_for,
copy_current_request_context
from random import random
from time import sleep
from threading import Thread, Event
import chess
import chess.uci
import chess.svg
import sys
```

Stockfish 10 download locally [8].

Results

Our team felt great success at the presentation day at the project day event. We have been chosen as the second favorite project by visitors QR-voting, and the system didn't fail to surprise mentors, friends and even people who don't know chess or played slightly too-aggressively with the pieces (still accuracy was shown).

The algorithm flow found to be very fast and responsive (if there is a delay in some part, is a delay we put for safety measures or best accuracy, for e.g. how much time we give the engine to think and give us a move suggestion).

We gained approximal 100% accuracy in the hand recognition part (determine the chess-engine skill level), 93% accuracy on piece classification, 100% on empty classification and the most important number – after all the logic layers we didn't encounter any issue with detecting the right move that was played.

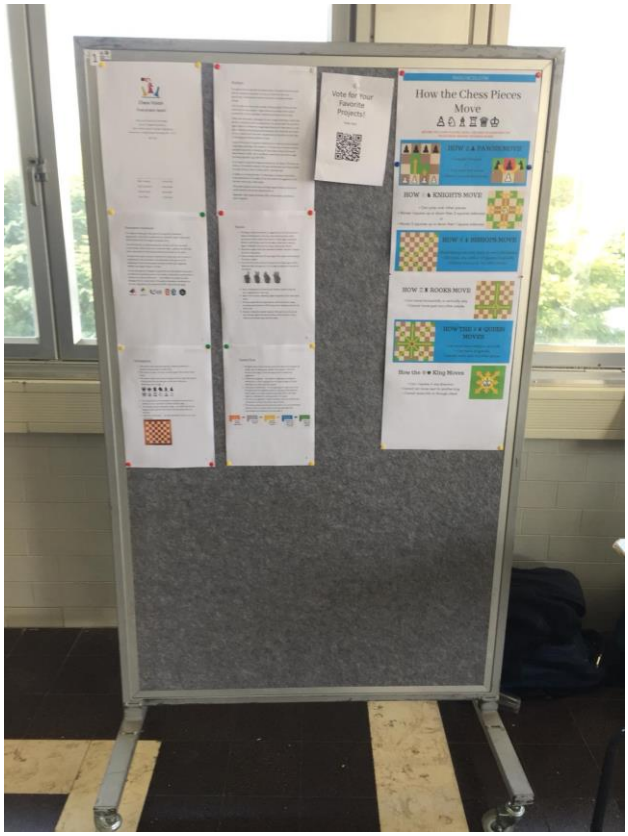


Figure 6.1 – presentation setup

אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

Figure 6.2 – presentation setup

YouTube video that present the project – with illegal move feedback, capturing, castling & winning:

Conclusions

Concluding remarks

This project gave us amazing opportunity to demonstrate creativity using the image-processing tools we learnt at class & assignments. In addition, we got to work with multi-platform code, communication protocols, microframework design, classification and much more.

Although we feel that we achieved any pre-set goal that we have, there is always room to improvements, especially if we are thinking about generalize the chess pieces. Future work can be done to implementing ideas from research's papers about classification of 3D chess pieces (not just with the image-processing tools that we learned, and even considering deep-learning for better results).

In the end, we are very proud in the project and the code we wrote. The microframework part can actually be use by anyone who wants a working tool to communicate with a chess engine & have a simple UI to show the results live.

Thank you,

Chess Vision team.

אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

References

1. Cheryl Danner & Mai Kafafy. *Visual Chess Recognition*. Stanford University

2. A Czyzewski, Maciej & Laskowski, Artur & Wasik, Szymon. (2018). *Chessboard and chess piece recognition with the support of neural networks*

3. S. Kolkur1, D. Kalbande2, P. Shimpi2, C. Bapat2, and J. Jatakia2. (2017). *Human Skin Detection Using RGB, HSV and YCbCr Color Models*. Published by Atlantis Press.

4. Gaurav Jain. (2010). *Detects skin by producing a map of "skin-like" pixels within a given image.* Retrieved at *https://www.mathworks.com/matlabcentral/fileexchange/28565-skin-detection.* (2018).

5. *Python-chess library*. *https://python-chess.readthedocs.io/en/latest/index.html*

6. *Flask framework in python.* *http://flask.pocoo.org/*

7. *Flask socketIO service*. *https://flask-socketio.readthedocs.io/en/latest/*

8. *Stockfish 10 – open source chess engine.* *https://stockfishchess.org/*

9. *UCI - Universal Chess Interface.* *https://en.wikipedia.org/wiki/Universal_Chess_Interface*

10. *Python 3.7.0.* *https://www.python.org/downloads/release/python-370/*

11. *Vision function – detectcheckboardPoints in MATLAB,* *https://www.mathworks.com/help/vision/ref/detectcheckerboardpoints.html*

12. Geiger, A., F. Moosmann, O. Car, and B. Schuster. "Automatic Camera and Range Sensor Calibration using a Single Shot," International Conference on Robotics and Automation (ICRA), St. Paul, USA, 2012.

אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

13. *Speeded up robust features*. Wikipedia
https://en.wikipedia.org/wiki/Speeded_up_robust_features

14. *MATLAB-Image Category Classification Using Bag of Features*.
https://www.mathworks.com/help/vision/examples/image-category-classification-using-bag-of-features.html

15. *MATLAB-Image Category Classification Using Deep Learning*.
https://www.mathworks.com/help/vision/examples/image-category-classification-using-deep-learning.html

16. *Support-vector machine*. Wikipedia.
https://en.wikipedia.org/wiki/Support-vector_machine