VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

LOKALIZACE ŠACHOVÝCH FIGUREK NA HRACÍ PLOŠE Z FOTOGRAFIE

VISUAL LOCALIZATION OF CHESS PIECES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **TOMÁŠ HAMPL**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MICHAL HRADIŠ, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Hampl Tomáš**
Program: Informační technologie
Název: **Lokalizace šachových figurek na hrací ploše z fotografie**
Visual Localization of Chess Pieces
Kategorie: Zpracování obrazu

Zadání:

1. Seznamte se základními metodami zpracování obrazu a metodami pro detekci objektů.
2. Vytvořte si přehled o současných přístupech pro vizuální analýzu stavu šachové partie a podobných her a současných přístupech pro 3D lokalizaci objektů z fotografie.
3. Připravte si datovou sadu vhodnou pro experimenty.
4. Vyberte vhodné metody pro lokalizaci šachovnice a šachových figurek z fotografie.
5. Implementujte vybrané metody a přístupy.
6. Proveďte experimenty nad datovou sadou a vyhodnoťte vlastnosti navrženého systému.
7. Porovnejte dosažené výsledky a diskutujte možnosti budoucího vývoje.
8. Vytvořte stručné video a demo aplikaci prezentující vaši práci, její cíle a výsledky.

Literatura:

- I. Szentandrasi et al.: Uniform Marker Fields: Camera localization by orientable De Bruijn tori. In ISMAR, 2012.
- He et al.: Mask R-CNN, in ICCV, 2017.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Hradiš Michal, Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 30. října 2020

Abstrakt

Hlavním cílem této práce bylo analyzovat stav šachové partie a lokalizovat šachové figurky na šachovnici. Rozpoznání šachovnice je založené na lokalizaci čar v obraze pomocí Houghovy transformace a PClines. Figurky byly detekovány modely konvolučních neuronových sítí - YOLOv3, YOLOv4 a YOLOv4 tiny. Vyhodnocení bylo provedeno na vlastní datové sadě. Detekce šachovnice dosahuje přesnosti 97%, kompletní lokalizace stavu šachovnice dosahuje 74,5% a lokalizace figurek 96%.

Abstract

The main goal of this thesis was to analyze state of the chess game and to locate chess pieces on the chessboard. Chessboard recognition is based on locating lines in image using Hough transform and PClines. The figures were detected by models of convolutional neural networks - YOLOv3, YOLOv4 and YOLOv4 tiny. Evaluation was perfomed on our data set. Chessboard detection achieves accuracy of 97%, complete localization of the state reaches 74,5% and piece localization 96%.

Klíčová slova

kalibrace kamery, PnP, homografie, detekce hran, Cannyho hranový detektor, Houghova transforamce, paralelní souřadnice, PClines, konvoluční neuronová síť, YOLO, detekce šachovnice, detekce figurek

Keywords

camera calibration, PnP, homography, edge detection, Canny edge detector, Hough transform, parallel coordinates, PClines, convolutional neural network, YOLO, chessboard detection, figure detection

Citace

HAMPL, Tomáš. *Lokalizace šachových figurek na hrací ploše z fotografie.* Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Hradiš, Ph.D.

Lokalizace šachových figurek na hrací ploše z fotografie

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Hradiše, Ph.D. a prohlašuji, že jsem uvedl všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Tomáš Hampl
11. května 2021

Poděkování

Tímto bych chtěl poděkovat panu Ing. Michalovi Hradišovi, Ph.D. za cenné rady a odbornou pomoc při vypracovávání této práce. Také bych chtěl poděkovat všem, kteří se podíleli na tvorbě datové sady. Obzvláště bych chtěl poděkovat Evě Michálkové a dále Sebastiánovi Gedeonovi za poskytnutí osvětlení.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Zaznamenávání tahů figurek při šachových partií je často náročným manuálním úkolem, který může hráčům bránit ve využití jejich plného potenciálu. Z tohoto důvodu se v profesionálních hrách využívají specializované šachové sety, které zaznamenávají automaticky jednotlivé tahy. Tyto šachové sety nejsou levnou záležitostí a právě proto se v posledních letech mnoho lidí pokusilo o vyřešení tohoto problému pomocí algoritmů počítačového vidění.

Předešlá řešení často vytvářela, se snahou zlepšit výsledky, fyzická omezení pro vzhled šachovnice a nebo její pozadí. Hlavní motivací této práce byla snaha tato omezení odstranit a vytvořit program, který by nevyžadoval prosté pozadí nebo jinak zbarvenou šachovnici, a stále dosahoval slušných výsledků.

Cílem práce je lokalizovat šachové figurky na šachovnici pomocí počítačového vidění. Tento problém můžeme rozdělit do dvou hlavních částí. Detekce šachovnice je založená na vědomosti, že šachovnice je jasně určena 18 čárami. K detekci čar se využívá dvou přístupů. První přístup je založen na Houghově transformaci a druhý na její modifikaci - *PClines*. Detekce figurek je uskutečněna pomocí konvoluční neuronové sítě. Vzhledem k záměrům práce byla vytvořena jednoduchá demo aplikace prezentující výsledky.

Druhá kapitola se věnuje základnímu principu problému a rozboru předešlých řešení, jejich nedostatků a výsledků. Třetí kapitola představuje hlavní část teoretické části. Zabývá se principy a metodami práce s obrazem a zvolenými metodami k detekci šachovnice a šachových figurek. Popisuje jejich principy a způsoby, jakými fungují. Čtvrtá kapitola pojednává o implementaci jednotlivých modulů, které jsou využity k řešení problému. Popisuje postupy jednotlivých programů k detekci šachovnice. Zároveň se věnuje způsobu trénování modelů neuronových sítí. Pátá kapitola se věnuje obsahu datové sady a způsobu tvorby anotací pro neuronovou síť. Šestá kapitola popisuje tvorbu a funkce vytvořené demo aplikace reprezentující výsledky. Poslední kapitola diskutuje výsledky jednotlivých modulů a porovnává je. Zároveň popisuje experimenty, které vedly k získání ideálních parametrů pro detektory čar.

Kapitola 2

Definice problému

Detekce a sledování šachových partií je známý a zajímavý problém, kterému se věnovalo a věnuje mnoho výzkumníků i fandů počítačového vidění. Digitální implementaci šachu je možno snadno automaticky zaznamenávat, ale fyzickou podobu šachu ne. Existují specializované šachové sety, které dokážou automaticky zaznamenávat šachové partie v reálném čase, ale cena těchto setů se může pohybovat v řádech stovek dolarů. Tyto šachové sety často využívají elektrických obvodů a magnetů pro lokalizaci nebo samotný pohyb šachovými figurkami. Ačkoliv je řešení tohoto problému pomocí zpracování obrazu technicky náročnější, je to jediný způsob, jak se vyhnout specializovanému setu. Mimoto detekce šachovnice a šachových figur je vitálním krokem pro stavbu robotů hrajících šachy, neboť se strategie hry odvíjí od rozpoznání a lokalizace vlastních a oponentových figurek.

Cílem tohoto projektu je analyzovat stav šachové partie z fotografie a vytvořit dvourozměrnou reprezentaci pro mnoho variací šachových setů. Tento problém můžeme rozdělit do tří hlavních částí a to následovných: detekce šachovnice, detekce a klasifikace šachových figurek a následná vizualizace výsledné detekce.

Přístupy k detekci šachovnice

Metody pro detekci šachovnice se často kategorizují mezi metody založené na detekci rohů a čar. Přístupy založené na detekci rohů fungují jen v případě, že nedochází k obstrukci jednotlivých rohů. Zároveň se předpokládá, že pozadí šachovnice je prosté. Aby detekce šachovnice pomocí této metody byla úspěšná, musí být prováděna před zahájením hry (rozestavením šachových figurek) [16] nebo pohled kamery musí být shora kolmo k šachovnici [27]. Tyto přístupy fungují dobře pro obecné potřeby zaznamenávání šachových partií, kde šachovnice může být detekována před začátkem hry bez šachových figurek. Přístup detekce rohů selhává pod obstrukcí rohů a velkém množství šumu v pozadí nebo šumu vytvořeného šachovými figurkami.

Přístupy založené na detekci čar jsou závislé na zpracování hran ve vstupním obrazu k identifikaci čar [7]. Vycházejí z předešlé vědomosti, jako je například skutečnost, že šachovou desku lze identifikovat dvojicí devíti čar, které jsou k sobě ortogonální. Tento přístup je také více robustní, protože se dobře vypořádává se šumem v obraze. Existují algoritmy, které se dokážou vypořádat i s obstrukcí figurkami. Ani tato metoda ovšem není dokonalá a může dojít k chybné detekci.

V jistých případech se k detekci šachovnice využívá obou metod zároveň. V této kombinaci se detekce rohů využívá ke zlepšení výsledků detekce. Nevýhodou této kombinace

je nutnost se vypořádat s velkým množstvím detekovaných rohů, které nejsou relevantní k šachovnici [7].

Přístupy k detekci šachových figurek

Metodám pro detekci šachových figurek nebylo věnováno tolik pozornosti. Aplikace, které se využívají ke sledování hry, předpokládají počáteční stav a následně využívají rozdílů intenzit ve snímcích před a po tahu pro sledování pohybů [16]. Metody, které nepředpokládají počáteční pozice šachových figurek, často využívají segmentace obrazu a tvarových deskriptorů k jejich identifikaci [7].

Segmentace obrazu ve většině případů závisí na schopnosti oddělit hlavní barvy šachovnice (světlá a tmavá políčka a figurky). V mnoho případech je obtížné rozlišit stejně barevné políčko a figurku a dochází k zavádění omezení, jako je například použití červenozelené šachovnice [7]. Dalším problémem je nutnost vytvořit referenční datovou sadu, ke které se detekované figurky přirovnávají, aby došlo k identifikaci. Tento přístup je náročný a referenční datová sada musí obsahovat pouze jednoznačné tvary. Dalším omezením je samotný úhel, pod kterým lze správně identifikovat šachovou figurku. Obecné řešení by mělo být schopné provádět rozpoznání šachových figurek na standardní nemodifikované sadě. Z tohoto důvodu se začalo využívat klasifikátorů s deskriptory a následně konvolučních neuronových sítí.

2.1 Existující řešení

Podkapitola se věnuje rozboru publikovaných článků, které se zaobíraly podobnou tématikou. Diskutuje jejich omezení, nevýhody, výsledky a způsob jakým přistoupily k řešení tohoto problému.

Automatic Chessboard Detecion for Intrinsic and Extrinsic Camera Parameter Calibration

Autoři článku využívají kombinaci přístupů založených na detekci rohů a čar [13]. Využívají Harrisova rohového detektoru v kombinaci s Houghovou transformací. Jediným problémem bylo velké množství detekovaných rohů, které nebyly relevantní k samotné šachovnici. Tato práce se nezabývá detekcí, klasifikací ani lokalizací šachových figurek a nebere v potaz obstrukci šachovnice. Z tohoto důvodu se s ní nebudeme dále zabývat.

Visual Chess Recognition

V tomto článku ze Stanfordské univerzity jeho autoři využívají k detekci šachovnice přístup založený na detekci čar [7]. Prvně detekují hrany a následně použijí Houghovu transformaci k lokalizování čar. Detektor hran se využívá ke zlepšení výsledků a urychlení výpočtu. K určení počátečního políčka využijí dvou dvojic detekovaných čar (sobě paralelních). Pomocí prvotního políčka dále lokalizují ostatní využitím homografie. K detekci šachových figurek využívají kumulativní úhlovou funkci pro popis tvaru spolu s Fourierovými deskriptory. Fourierovy deskriptory jsou vypočítány pro každou referenci v datové sadě, kde prvotně použijí kumulativní úhlovou funkci k získání popisu tvaru a následně vypočítají normalizované Fourierovy koeficienty. Samotná detekce probíhá jako porovnávání referenčních

obrázků k nejbližší shodě. Pří prvotním testování zjistili, že segmentování původních fotografií bylo extrémně náročné. Aby se s tímto problémem vypořádali, zaměnili světlá a tmavá políčka za červená a zelená. Tím sice zlepšili detekci šachovnice a segmentaci figurek, ale zároveň vytvořili omezení v podobě používání červenozelené šachovnice. Detekce figurek je invariantní a funguje jen k předem připravené datové sadě nebo dosti podobným figurkám pod stejnými úhly. Vzhledem k barvě šachovnice a k tomu, že existuje velké množství variací šachových figurek, tento přístup není ideální pro obecné řešení.

ChessVision: Chess Board and Piece recognition

Jialin Ding ze Stanfordské univerzity, se v jeho práci zaměřil spíše na detekci šachových figurek [9]. Původní program *ChessVision* byl implementován pomocí přístupu detekce čar, ale tato technika měla značnou chybovost z důvodu šumu v obraze a často nebyla schopna správně detekovat šachovou desku. Z tohoto důvodu a ze samotného faktu, že práce byla zaměřena na detekci šachových figurek, se autor rozhodl zjednodušit lokalizaci šachovnice přidáním minimálního počtu interakcí s uživatelem. Uživatel je na začátku běhu programu prezentován vstupním obrázkem, kde označí rohy šachovnice ve směru hodinových ručiček (začátkem byl levý horní roh). Následně je použita projektivní transformace pro získání pohledu shora. Detekce šachových figurek byla uskutečněna pomocí klasifikátorů. Byla vytvořena datová sada, ve které byly všechny reference v poměru stran 1:2 (64 x 128 pixelů). Z obrázků se extrahovaly příznaky pomocí transformace měřítkově nezávislých rysů (SIFT) a histogramu orientovaných gradientů (HOG). Byly použity SVM (z angl. *Support-vector machine* k trénování „one vs rest“ klasifikátorů pro každou třídu. Pozitivní tréninkový set se skládá z příznaků dané třídy a negativní tréninkový set se skládá ze všech ostatních příznaků. K detekci a klasifikaci figurek je použita modifikovaná technika posuvného okna, kdy jsou pro každé políčko spuštěny všechny klasifikátory s upravenou výškou okna podle jednotlivých tříd. Výstupem jsou matice pravděpodobnosti všech tříd pro všechna políčka. Porovnají se pravděpodobnosti tříd políčka a vybere se třída s nejvyšší pravděpodobností jako predikce pro dané políčko. Barva figurky se určí pomocí poměru černých a bílých pixelů v každém čtverečku. Tento přístup byl testován jen na jednom šachovém setu a nedá se vytvořit úplný závěr, zda je dost robustní a jak by si tento přístup poradil s variancí šachových figurek. Výsledek detekce je 95% a klasifikace 85% pomocí HOG příznaků.

Chess Recognition Using Computer Vision

Ramani Varun a Sukrit Gupta publikovali metodu doplňující stávající výzkum pomocí shlukování k segmentování šachovnice a dílků bez ohledu na barevné schéma [43]. Pro rozpoznávání šachových figurek metoda zavádí nový přístup pomocí regionální konvoluční neuronové sítě k trénování robustního klasifikátoru. V jejich přístupu pracují s předpokladem, že pozadí šachovnice je prosté. Využívají Cannyho hranový detektor k získání hran z předzpracovaného obrazu, ze kterého detekují čáry pomocí Houghovy transformace. Díky bílému pozadí obrazu získají přesnou polohu šachovnice. Aby získali pohled shora na šachovnici, využijí projektivní transformace. Po získání pohledu shora, začnou rozpoznávat a segmentovat hrací desku pomocí shlukování barev. Segmentace je důležitá pro odlišení šachovnice a šachových figurek. Pro odlišení využívají algoritmu *k-means*. Shluk obsahující šachové figurky zpracovávají pomocí prahové operace k binarizaci obrazu. Následně použijí morfologický operátor pro vytvoření minimálních ohraničujících boxů pro skvrny a spustí

na ně lineární SVM, který jednotlivé skvrny klasifikuje. Klasifikátor byl trénován neuronovou sítí *Alexnet*. Datová sada obsahuje 100 snímků každé figurky pro obě barvy. Tento přístup dosáhl přesnosti 69% na jejich datové sadě. Problémy nastaly s fotkami pod nižšími úhly od pohledu shora, kdy segmentace vytvořila box přes více figurek. Dalším problémem bylo překrývání figurek a následná chyba rozpoznání.

Chessboard and Chess Piece Recognition With the Support of NeuralNetworks

Za zmínku také stojí práce autorů Maciej A. Czyzewski, Artur Laskowski a Szymon Wasik, kteří se zaměřují na co nejefektivnější řešení detekce šachovnice a rozpoznání figurek [8]. Jejich řešení je odolné špatným světelným podmínkám a úhlu zachycení šachovnice. Zároveň dosahuje dobrých výsledků s velkou různorodostí šachových setů. Algoritmus funguje iterativně a dá se rozdělit do tří hlavních částí: detekce čar, hledání mřížkových bodů a umístění šachovnice. Následně jsou detekovány figurky a vytvořen řetězec FEN.

2.2 Šachy

Šachy nebo taktéž šach je rekreační a kompetitivní strategická desková hra pro dva hráče. Hra spočívá v pohybu jednotlivými kameny (tyto kameny nazýváme figurky) po šachovnici. Pro figurky platí jasná pravidla. Cílem šachové partie je dát soupeři mat. Mat představuje situaci, kdy dojde k napadení soupeřova krále takovým způsobem, že ze soupeřovy strany není možné této situaci zabránit jak pohybem krále, tak pohybem jiné figurky. Samotná hra se skládá z hrací desky – šachovnice, která má 64 políček a 32 figurek, které jsou rozděleny barvou na 16 bílých a 16 černých figurek.

Hrací deska šachu

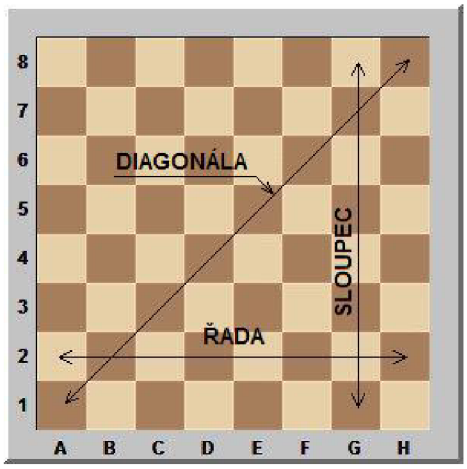
Šachovnice [38] je čtvercového tvaru a je pravidelně rozdělená na políčka v 8 řadách a 8 sloupcích. Pole jsou střídavě zbarvena bíle a černě (označujeme je jako pole světlá a tmavá). Na obrázku 2.1 je vyobrazen vzhled šachovnice. Šachovnice je umístěna mezi hráči takovým způsobem, že každý z hráčů má po své levé ruce rohové černé políčko a po své pravé ruce rohové bílé políčko.

Figurky šachu

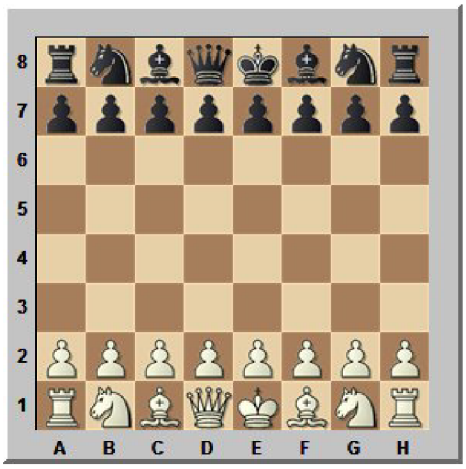
Na začátku hry má každý z dvojice hráčů celkem šestnáct figurek šesti druhů [39]. Jsou to král, dáma, dvě věže, dva střelci, dva jezdci a osm pěšců (figurky jsou vyjmenovány podle důležitosti ve hře). Hráč s bílými figurkami je označován jako bílý a soupeř jako černý. Na obrázku 2.2 je zobrazen počáteční stav šachové partie v dvourozměrné reprezentaci.

Notace šachu

Notace šachu [6] slouží k zaznamenání jednotlivých tahů šachové partie. K zápisu jednotlivých tahů se využívá algebraické notace, která označuje číslo tahu, název figurky a na jakou pozici se figurka pohne. Každá figurka kromě pěšáka má svou zkratku a každé políčko na šachovnici má svůj identifikátor. Sloupce šachovnice jsou označeny písmeny A až H a řádky čísly 1 až 8. Na figurky se odkazuje pomocí jejich zkratek, které jsou zapsány



Obrázek 2.1: Notace šachovnice.
Převzato z [38].



Obrázek 2.2: Základní rozestavení šachovnice.
Převzato z [39].

Obrázek 2.3: Ukázka bílých figurek.
Převzato z [26].

Obrázek 2.4: Ukázka černých figurek.
Převzato z [26].

velkými písmeny a to Král – K, dáma – Q, věž – R, střelec – B a jezdec – N [31]. Pěšec je jediná figurka, která nemá svou zkratku a pokud je s ním proveden pohyb, zapíše se jen destinace figurky. Některé tahy (zaujmutí, rošáda, šach, mat, proměna pěšce a remíza) se nedají zapsat algebraickou notací a mají své vlastní speciální symboly.

Forsyth-Edwards notace

Standardní notací šachu pro popis aktuálního stavu šachové partie se nazývá Forsyth-Edwards Notation (dále jen FEN). [12] Tato notace se využívá k zaznamenání všech potřebných informací pro pokračování šachové partie z určeného místa. FEN je jednořádkový zápis složený z ASCII charakterů rozdělený do šesti částí a to: pozice figurek, barva hráče na tahu, možnost rošády, En passant, pravidlo padesáti tahů a celkový počet tahů. Pozice figurek je popsána po jednotlivých řádcích, jak jsou uspořádány na šachovnici z pohledu bílého hráče. Na figurky se odkazuje se stejnými zkratkami jako v algebraické notaci, ale zde má i pěšák svou zkratku - P. Figurky se odlišují velikostí písmen. Černé figurky jsou zapsány malými písmeny a bílé velkými. Počet prázdných míst je zapsán číslem mezi stranami a figurkami. Řetězec 2.1 je ukázkou startovací pozice šachové partie, která je zobrazena na obrázku 2.2 v notaci Forsyth-Edwards.

$$rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNRwKQkq - 01 \tag{2.1}$$

Kapitola 3

Metody

Kapitola je rozdělena do tří částí. První část popisuje terminologii a teoretické poznatky transformace bodů z reálného světa do roviny obrazu a odhad pozice kamery v obraze. Druhá část popisuje přístupy k detekci čar a nutné předzpracování obrazu k získání dobrých výsledků. Třetí část kapitoly popisuje princip konvolučních neuronových sítí a následně se zaměří na modely YOLO.

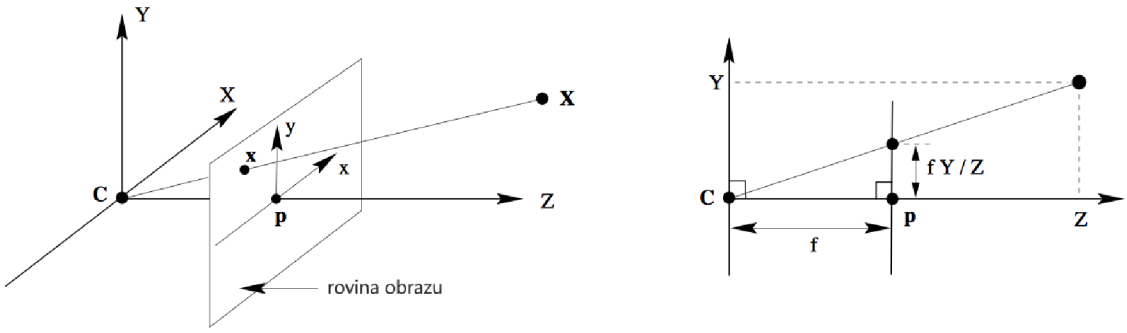
3.1 Transformace obrazu

Podkapitola popisuje základní terminologii a princip transformace bodů z reálného světa do obrazového světa, získávání projekční matice kamery a řešení PnP problému.

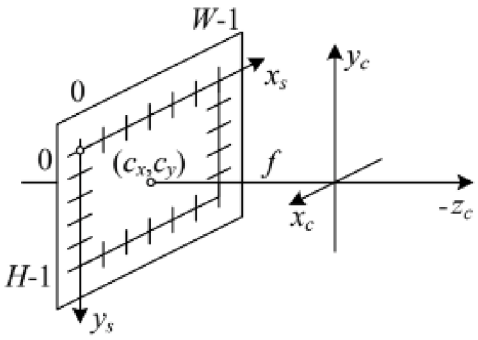
Perspektivní promítání

Perspektivní projekce je popsaná geometrickým modelem dírkové kamery (z angl. *pinhole camera model*) [17]. Vzniklá projekce reprezentuje tří dimenzionální objekt z reálného světa promítnutý do dvou dimenzionální roviny obrazu. Tato projekce je charakteristická deformací promítaných objektů. Výsledek je takový, že objekty s větší vzdáleností od kamery se zdají menší a vice versa.

Na obrázku 3.1 bod C představuje střed kamery, p představuje střed obrazu, f představuje ohniskovou vzdálenost a X představuje bod v reálném světě s homogenními souřadnicemi $[X, Y, Z]^T$, který se promítá do bodu x se souřadnicemi $[x, y]^T$, podle vztahu



Obrázek 3.1: Model dírkové kamery. Převzato z [23].



Obrázek 3.2: Zjednodušená vnitřní matice kamery. Převzato z [41].

$$x' = \frac{xf}{z}, \; y' = \frac{yf}{z}, \tag{3.1}$$

kde x', y' jsou souřadnice promítaného bodu a x, y jsou souřadnice bodu ve scéně.

Kalibrace kamery

Kalibrace kamery je proces zjišťování vnitřních a vnějších parametrů kamery [41]. Tyto parametry se následně využívají ke korekci zkreslení obrazu a mapování bodů z trojrozměrného reálného světa do dvourozměrného obrazového světa zachyceného kamerou. Správná kalibrace kamery je základním kamenem pro jakákoliv měření a vykreslování obrazců v obrazové rovině.

Kalibrace kamery je v této práci velmi důležitá pro tvorbu ohraničujících rámečků figurek. Každá šachovnice má anotované rohy. Na základě těchto informací dokážeme vypočítat matici kamery a koeficienty zkreslení. Tyto parametry potřebujeme následně pro vyřešení problému odhadu pozice.

Na obrázku 3.2 je zobrazena zjednodušená vnitřní matice kamery, kde f je ohnisková vzdálenost, (cx, cy) je střed obrazu, W je šířka a H je výška obrazu. Mapování bodu z reálného světa do obrazového světa můžeme popsat rovnicí

$$\tilde{X}_s = K\left[R\,t\right]p_w = Pp_w, \tag{3.2}$$

kde $\tilde{X}_s$ představuje homogenní souřadnice 2D bodu v obraze, K vnitřní matici kamery, $[Rt]$ vnější matici kamery, P_w homogenní souřadnice 3D bodu v reálném světě a P projekční matici kamery.

Vnitřní matice kamery představuje vnitřní parametry (ohnisková vzdálenost, optické centrum a koeficienty radiálního zkreslení čočky kamery).

$$K = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.3}$$

kde f_x a f_y reprezentují ohniskovou vzdálenost v osách x a y kamery, γ představuje koeficient šikmosti a parametry c_x a c_y představují souřadnice optického středu. Většina současných kamer využívá stejné ohniskové vzdálenosti (tedy $f_x = f_y$), které jsou navíc na sebe kolmé.

Vnější matice kamery obsahuje vnější parametry, které představují orientaci kamery vzhledem ke světovému souřadnicovému systému. Tato matice je složená z rotační matice R a translačního vektoru t. Matice je typu 3x4 a je ve tvaru

$$\begin{bmatrix} R\,t \end{bmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{pmatrix}, \tag{3.4}$$

kde rotační matice R je typu 3×3 a translační vektor t je typu 3×1.

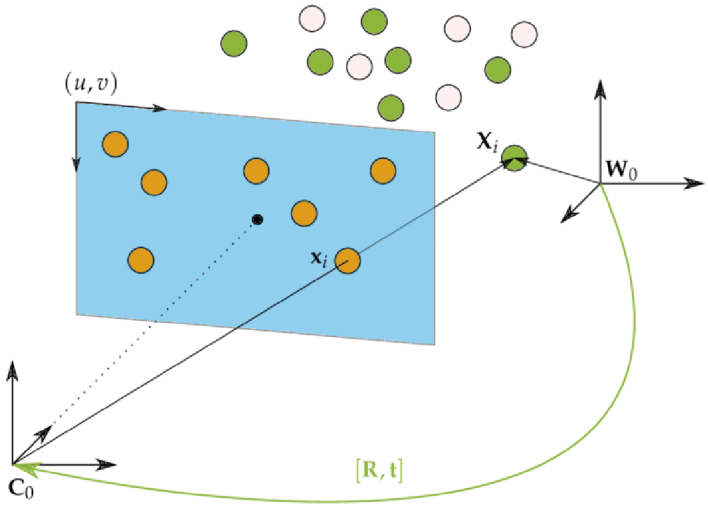
Kalibrace kamery se často provádí na šachovnicových vzorech [45], neboť jsou zřetelné a lehké na detekci v obraze. Rohy šachovnicových polí jsou ideální na detekci, protože mají ostré gradienty ve dvou směrech. Kromě toho se rohy nacházejí v průsečících šachovnicových čar. Po detekci rohů a získání jejich souřadnic v obraze se následně použijí algoritmy pro kalibraci kamery. Pro kvalitní kalibraci je potřeba zachytit vícero pohledů na šachovnicový vzor. Různé pohledy zaručí, že se do úvahy budou brát různé druhy zkreslení obrazu.

Problém Perspective-n-Point

Cílem *Perspective-n-Point* (zkratka PnP) [29] problému je určit relativní pozici a orientaci kamery vzhledem k jejím vnitřním parametrům a množině n korespondencí mezi trojrozměrnými body reálného světa a jejich dvourozměrnými projekcemi v obrazové rovině. Pozice kamery je složená z šesti stupňů volnosti (z angl. *six degrees of freedom*) a ty jsou rotační (otáčení, stoupání a vybočení) a translační. Z toho vyplývá, že je potřeba získat minimálně tři korespondenční páry bodů k vyřešení PnP problému. Většina dostupných řešení jsou aplikovatelné pro případy, kdy je známá informace více jak tří korespondenčních párů bodů. Tento problém má mnoho uplatnění v oblasti počítačového vidění, robotiky a virtuální reality a získal velkou pozornost v těchto komunitách.

Algoritmus PnP může být použit k odhadu pozice [30]. Odhad pozice využívá tato práce k vykreslení ohraničujících boxů pro figurky. Díky tomu, že figurka na šachovnici má jasné umístění, dokážeme získat korespondenční body a vypočítat pozici kamery. Následně můžeme využít tuto pozici k vykreslení kvádru okolo daného políčka, na kterém se figurka nachází.

Na obrázku 3.3 je zobrazený princip perspektivního promítání oranžových bodů v reálném světě do žlutých bodů obrazové roviny. Bod P_i z reálného světa je promítán do bodu roviny obrazu u_i. W reprezentuje světový souřadnicový systém, c kamerový souřadnicový systém, f ohniskovou vzdálenost, R rotační matici a t translační vektor.



Obrázek 3.3: Princip perspektivního promítání bodů. Převzato z [33].

Body vyjádřené v reálném světě X_w jsou promítány do obrazového rámce $[u, v]$ modelem perspektivní projekce a vnitřní matice kamery následovně

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K\Pi^c T_w \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix},$$
$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}, \tag{3.5}$$

kde K představuje matici vnitřních parametrů kamery, Π model perspektivní projekce a ${}^cT_w$ představuje rotační a translační vektory.

Odhadovaná pozice se skládá z rotačních a translačních vektorů, které umožňují transformaci tří dimenzionálního bodu vyjádřeného ve světovém rámci do dvou dimenzionálního bodu obrazové roviny:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = {}^c T_w \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix},$$
$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}, \tag{3.6}$$

V dnešní době existuje několik různých přístupů k problému PnP. Knihovna *OpenCV* některé z nich implementuje, jsou to:

- **Iterative** - Iterativní metoda je založena na Levenberg-Marquardt optimalizaci. Funkce hledá polohu kamery takovou, která minimalizuje chybu opakované projekce. Minimální počet bodů pro výpočet je tři.
- **P3P** - Metoda je založena na článku *Complete Solution Classification for the Perspective-Three-Point Problem*. V tomto případě funkce vyžaduje přesně čtyři objektové a obrazové body.
- **UPNP** - Metoda je založena na publikaci *Exhaustive Linearization for Robust Camera Pose and Focal Length Estimation*. Funkce odhaduje parametry fokální vzdálenosti s předpokladem, že jsou oba parametry stejné. Následovně aktualizuje matici kamery s odhadovanými parametry.
- **IPPE** - metoda vychází z článku *Infinitesimal Plane-Based Pose Estimation*. Metoda vyžaduje koplanární objektové body a alespoň čtyři body.
- **IPPE SQUARE** - metoda je založena na článku *Infinitesimal Plane-Based Pose Estimation*. Tato metoda je vhodná pro odhad pozice markeru.

Homografie

Projektivní lineární transformace neboli homografie [10] je invertibilní mapování bodů a přímek na projektivní rovině P^2. Jinými slovy, pokud máme homogenní souřadnice bodu a v rovině π a souřadnice bodu a' v rovině π' takové

$$\begin{aligned} a &= [x, y, w]^T \\ a' &= [x, y, w]^T \end{aligned} \tag{3.7}$$

kde w představuje váhu bodu, potom homografie představuje perspektivní transformaci mezi rovinami π a π'. Tato transformace může být vyjádřena vztahem

$$a' = Ha, \tag{3.8}$$

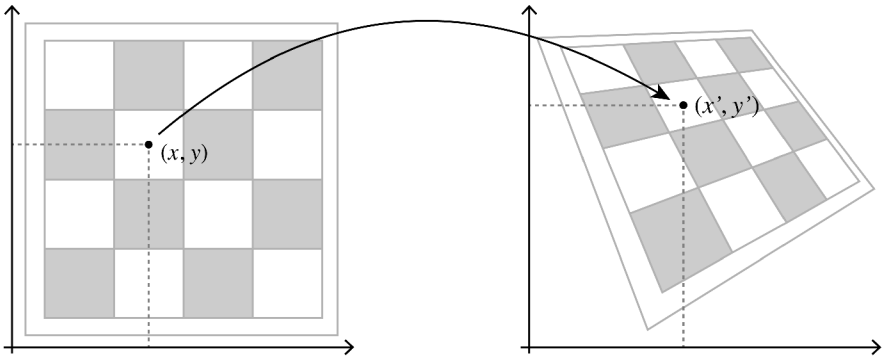
kde H představuje transformační matici homografie o rozměrech 3×3. Transformační matice je definována jako

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}, \tag{3.9}$$

kde h_{33} má hodnotu 1 nebo se vypočítá rovnicí

$$h_{11}^2 + h_{12}^2 + h_{13}^2 + h_{21}^2 + h_{22}^2 + h_{23}^2 + h_{31}^2 + h_{32}^2 + h_{33}^2 = 1. \tag{3.10}$$

Pro získání transformační matice homografie je prvně potřeba nalézt minimálně osm bodů (čtyři body v každém zobrazení). V praxi se využívá bodů více pro snížení chyby výpočtu. Tyto body se následně spojí do odpovídajících dvojic a jsou pro ně vytvořeny jednotlivé lineární rovnice, které vyjadřují vztah objektu v jednom zobrazení a objektu v druhém zobrazení.



Obrázek 3.4: Promítnutí bodu homografií z destinačního obrázku (vlevo) do obrázku v reálném světě (vpravo). Převzato z [46].

Tato práce využívá homografie k určení souřadnic políčka, na kterém se nachází daná figurka. Vzhledem k tomu, že máme anotované rohy šachovnice a rohy referenčního obrázku, dokážeme vypočítat transformační matici homografie a následně převést body z referenčního obrázku do obrázku reálného pomocí inverzní matice homografie. Tímto získáme reálné souřadnice daného políčka. Transformace je zobrazena na obrázku 3.4, kde se promítá bod *(x, y)* z destinační obrázku do bodu *(x', y')* v reálném světě.

3.2 Detekce přímek

Detekce čar pomocí Houghovy transformace a paralelních souřadnic je základním kamenem této práce. Obě metody jsou závislé na detekci hran. Tato podkapitola popisuje jednotlivé metody, terminologii a způsob, jakým probíhá detekce. Zároveň popisuje jakým způsobem probíhá detekce hran.

Detekce hran

Detekce hran je stěžejní částí zpracování obrazu pro detekci čar. Jejím úkolem je extrahovat významné rysy obrazu, snížit výpočetní náročnost a zlepšit výsledek detekce čar. Detekce hran nám umožňuje zjistit polohu a vlastnosti jednotlivých čar.

Hrany jako takové jsou charakterizovány jako lokální změny intenzity v digitálním obraze [28]. Hranové detektory můžeme rozdělit na detektory využívající první derivaci a na detektory využívající druhou derivaci. Derivace se aproximují konvolucí malých okolí v obrazu. Konvoluční jádra se typicky volí jako matice o velikosti 3×3. Příkladem detektorů prvních derivací jsou *Sobelův*, *Robertsův* a *Prewittsův* operátor. Příkladem detektorů druhých derivací jsou *Marr-Hilderthův* operátor a *LoG*.

Cannyho hranový detektor

Cannyho detektor hran [14][5] je nejznámější hranový detektor. Byl navržen v roce 1986 Johnem F. Cannym. Poskytuje dobrou a spolehlivou detekci na úkor výpočetní náročnosti. Je to více krokový algoritmus, který byl založen na třech hlavních kritériích [3]:

1. **Kvalita detekce** (nízká chybovost): Detekce musí přesně zachytit co nejvíce hran zobrazených v obraze a zároveň nesmí označit falešné hrany způsobené šumem.

2. **Přesnost detekce:** Poloha nalezené hrany musí být co nejblíže poloze skutečné hrany v obraze.

3. **Jednoznačnost:** Daná hrana v obraze by měla být označena pouze jednou. Nesmí docházet k vícenásobnému označení jedné hrany.

Samotný průběh detektoru je rozdělen do čtyř částí [32] a jeho výsledek je zobrazen na obrázku 3.5:

1. **Redukce šumu:** První krok algoritmu je odstranění šumu v obraze. Na vstupní obraz se aplikuje Gaussův filtr pro vyhlazení obrazu.

2. **Určení velikosti gradientu:** Vyhlazený obraz je filtrován Sobelovým operátorem v horizontálním i vertikálním směru pro získání prvních derivací. Následně z těchto dvou derivací získáme velikost a směr gradientu pro každý pixel. Masky pro Sobelův filtr jsou následující

$$\begin{aligned} G_x &= \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \\ G_y &= \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, \end{aligned} \tag{3.11}$$

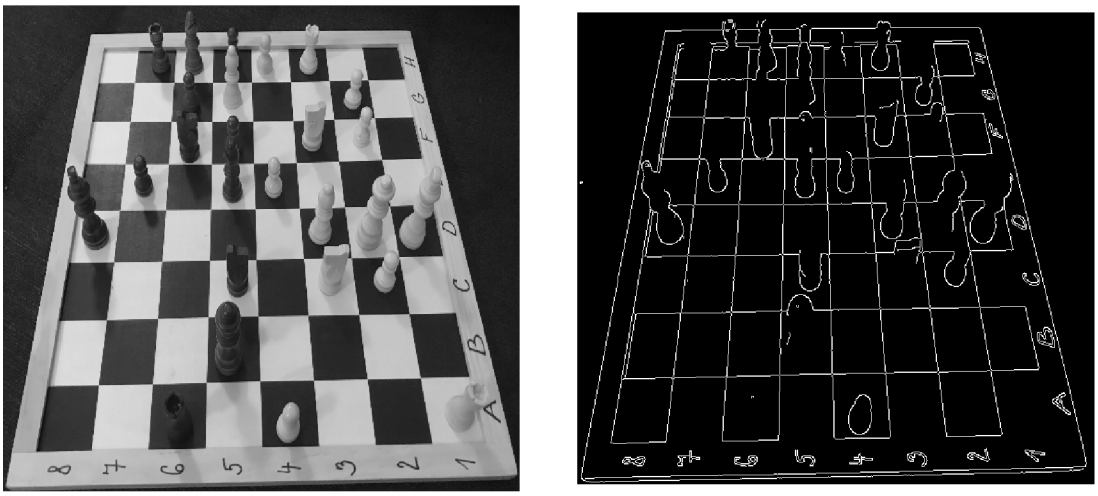
   kde G_x je maska pro detekci horizontálních hran a G_y je maska pro vertikální. Následně vypočítáme velikost gradientu a směr

$$\begin{aligned} G &= \sqrt{G_x^2 + G_y^2}, \\ \theta &= \arctan \frac{G_y}{G_x}, \end{aligned} \tag{3.12}$$

   kde G je velikost gradientu, θ je směr, G_x a G_y jsou výsledky použití Sobelových masek.

3. **Nalezení lokálních maxim** („non-maximal suppression“): Následně dochází ke ztenčení hran a odstranění nežádoucích bodů. Jsou vyhledána lokální maxima hran ve směrových sousedstvích a jsou utlumeny body, které nedosahují těchto hodnot.

4. **Prahování s hysterezí:** Posledním krokem algoritmu je eliminace nevýznamných hran. K určení, zda hrana je významná či ne, se využívá dvou prahů T_{max} a T_{min}. Pokud je hodnota gradientu vyšší jak T_{max}, dochází k označení za hranu významnou. Pokud je hodnota gradientu menší jak T_{min}, dochází k likvidaci hrany. Pokud se hodnota gradientu hrany nachází mezi prahy T_{max} a T_{min} a je spojená s hranou významnou, hrana je označena za hranu významnou, pokud není spojena s hranou významnou, je zlikvidována.



Obrázek 3.5: Ukázka aplikace Cannyho detektoru hran na původní obrázek.

Houghova transformace

Houghova transformace (zkráceně HT)[24][15] je technika zpracování obrazu, která se využívá k extrakci příznaků v obraze. Patří mezi nejčastěji používané metody na detekci jednoduchých geometrických objektů. Tato metoda vyžaduje, aby hledané objekty byly definovány v parametrické formě. Klasická Houghova transformace je běžně používaná k detekci pravidelných geometrický objektů, u kterých je parametrický popis dobře známý. Mezi tyto geometrické objekty patří přímka, kružnice, elipsa atd. Hlavní výhodou Houghovy transformace je to, že je tolerantní k nepravidelnostem a přerušení křivek. Zároveň je relativně neovlivněná šumem v obrazu.

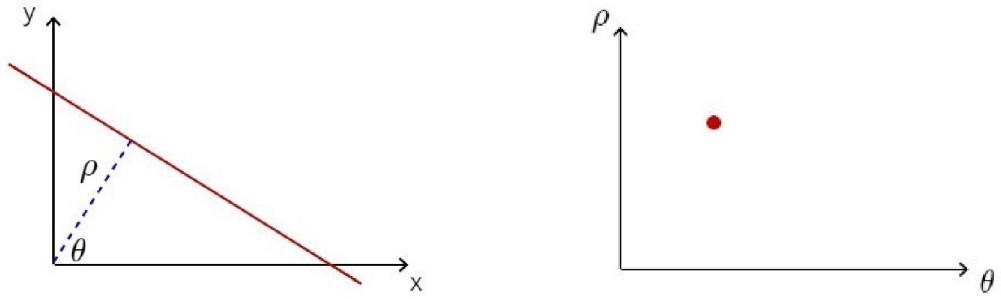
Vstupem této metody je ve většině případů obraz, který byl již předem zpracován, nejčastěji barevnou korekturou a následovně detektorem hran. Tento postup zvyšuje pravděpodobnost nalezení hledaných struktur a zároveň snižuje náročnost, a tím urychluje výpočet. Výstupem je soubor objektů v parametrické formě.

Detekce přímek je nejjednodušší případ [1] využití klasické Houghovy transformace. Samotná přímka představuje nejjednodušší objekt, který můžeme v obraze rozeznat. Jednoduchý objekt je takový, který můžeme reprezentovat malým počtem parametrů. Například přímka může být reprezentována dvěma parametry, a to jsou sklon přímky a průsečík přímky s osou y. Dalším příkladem může být kružnice, která je reprezentována třemi parametry. Vícero objektů (například přímek) může tvořit komplexnější objekty.

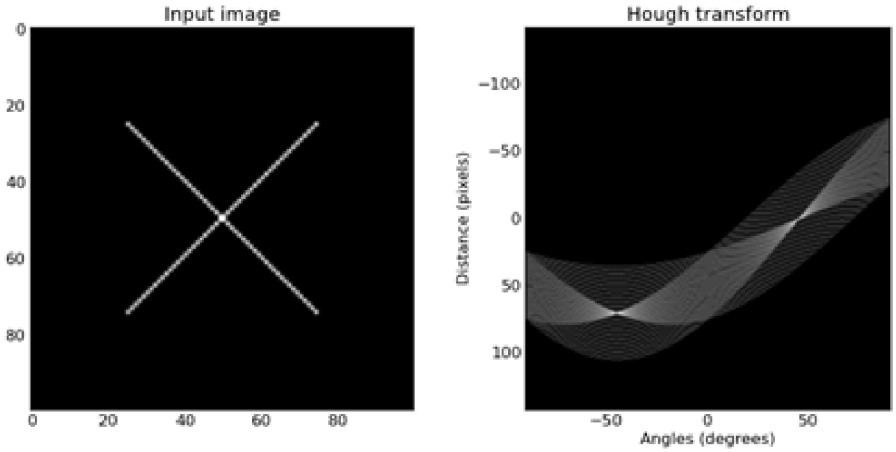
V první řadě je potřeba si parametricky vyjádřit hledaný objekt. Přímku můžeme vyjádřit v Polárním souřadnicovém systému následovně [4]

$$\begin{aligned} y &= (-\frac{\cos\theta}{\sin\theta}) \cdot x + (\frac{\rho}{\sin\theta}), \\ \rho &= x \cdot \cos\theta + y \cdot \sin\theta, \end{aligned} \tag{3.13}$$

kde ρ představuje kolmou vzdálenost přímky od počátku souřadnic k hledané přímce a θ je úhel, který svírá přímka s osou x viz. 3.6 vlevo. Další možností, jak popsat parametricky přímku, je v Kartézském souřadnicovém systému $y = mx + c$, kde m představuje sklon přímky a c průsečík přímky s osou y. Důvod, proč se nevyužívá tohoto vzorce je ten, že sklon pro vertikální a horizontální přímky může být v rozmezí $-\infty$ až ∞. Teoreticky tento



Obrázek 3.6: Transformace přímky z Eukleidovského prostoru do bodu v Houghově prostoru. Převzato z [22].



Obrázek 3.7: Promítnutí vstupního obrazu (vlevo) do Houghova prostoru (vpravo). Převzato z [42].

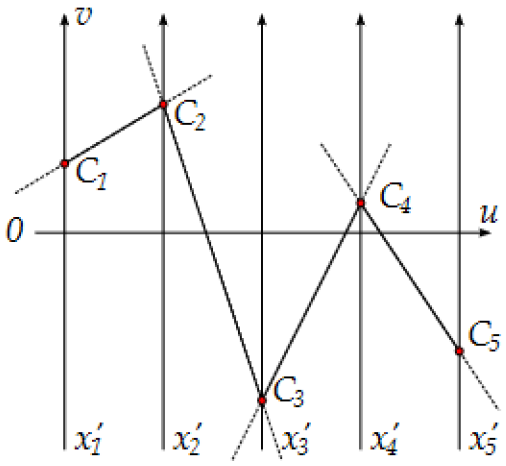
problém může nastat i u prvního vyjádření, kdy ρ by mohlo spadat do rozmezí 0 až $+\infty$. V praxi ale tento problém nastat nemůže, neboť velikost obrázku je vždy konečná.

Při detekci se pro každý obraz vytváří tzv. Houghův prostor (častěji nazýván akumulátor). Akumulátor představuje dvourozměrný prostor, kde osy představují vzdálenosti ρ a úhly θ. Každý bod tohoto prostoru nese informaci o jeho hodnotě intenzity. Převod bodu z Eukleidovského do Houghova prostoru je zobrazen na obrázku 3.6.

Detekce přímek

Prvním krokem algoritmu je vytvoření akumulátoru, jehož rozměry záleží na požadované přesnosti a rychlosti výpočtu. Pro každý bod obrazu se provede výpočet ρ dosazením souřadnic bodu do výše zmíněné rovnice a posouváním úhlu θ v rozmezí od 0 do π o předem určený krok. Tímto získáme pro určený bod souřadnice všech bodů v akumulátoru a zvýšíme jim intenzitu o předem dohodnutou hodnotu (nejčastěji o 1). Tento krok nám vytvoří pro každý bod v obraze sinusoidu v akumulátoru. Na konci tohoto kroku budeme mít naplněný akumulátor.

Na obrázku 3.7 je možno vidět, že se v akumulátoru vyskytují maxima (nejsvětlejší body). Maxima identifikují přímku v původním obraze. Vznikají protnutím většího počtu



Obrázek 3.8: Reprezentace pěti dimenzionálního vektoru v prostoru PC. Převzato z [11].

křivek a znamená to, že přímka představovaná následujícím maximem má více bodů v obraze. Tato maxima představují kandidáty na detekované přímky.

Paralelní souřadnice

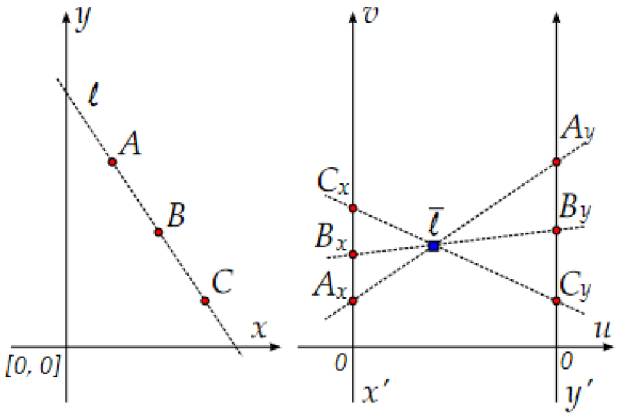
Detekce přímek v obraze se často provádí pomocí Houghovy transformace. Algoritmus PC-lines [11] je modifikací Houghovy transformace a je založen na paralelních souřadnicích. Paralelní souřadnice (zkratka PC) byly vynalezeny v roce 1885, kdy Maurice d'Ocagne popsal metodu transformace paralelních souřadnic [44]. Alfred Inselberg je popularizoval v roce 1985. V současné době se paralelní souřadnice používají pro vizualizaci a analýzu multidimenzionálních dat.

Paralelní souřadnicový systém reprezentuje vektorový prostor rovnoběžnými osami. Každý N-rozměrný vektor je reprezentován N-1 přímkami, které spojují jednotlivé osy. Obrázek 3.8 představuje ukázku pěti dimenzionálního prostoru. V daném prostoru se nachází vektor se souřadnicemi $[C_1, C_2, C_3, C_4, C_5]$. V systému paralelních souřadnic je vektor vyobrazen jeho hodnotami na jednotlivých osách, které jsou následně spojeny $N-1$ přímkami. Pro definování těchto bodů se využívá notace $(u, v, w)_{P^2}$ pro homogenní souřadnice v projektivním prostoru P^2.

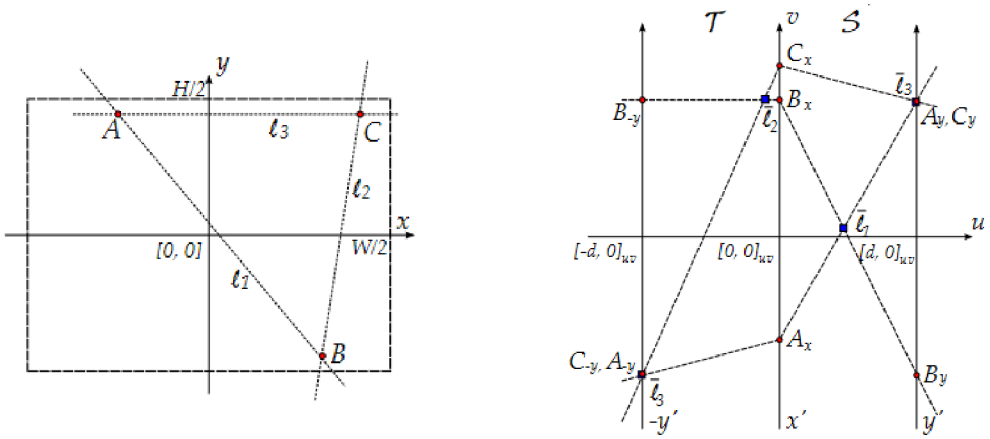
Ve dvourozměrném případě jsou body v Eukleidovském prostoru reprezentovány jako čáry v prostoru paralelních souřadnic. Jejich průsečíky následně tvoří reprezentaci přímky v prostoru paralelních souřadnic. Na základě tohoto vztahu, podobně jako v Houghově transformaci, je možno definovat mapování přímky do bodu v těchto prostorech. Na obrázku 3.9 můžeme vidět transformaci kolineárních bodů A, B a C v Eukleidovském prostoru do bodu $\bar{\ell}$ v prostoru paralelních souřadnic.

Detekce přímek

V Eukleidovském prostoru $x-y$ je přímka vyjádřena parametricky jako $\ell : y = mx + b$. V prostoru paralelních souřadnic $u-v$ je reprezentována trojicí $\bar{\ell} = (d, b, 1-m)_{P^2}$, kde d představuje vzdálenost paralelních os x' a y'. Reprezentace čáry $\bar{\ell}$ je mezi osami x' a y' pouze tehdy, když m je v intervalu $(-\infty, 0)$. Pro $m = 1$ reprezentace $\bar{\ell}$ představuje



Obrázek 3.9: Transformace přímky do dvourozměrného prostoru PC. Převzato z [11].



Obrázek 3.10: Transformace trojúhelníku z Eukleidovského prostoru (vlevo) do prostoru paralelních souřadnic (vpravo). Převzato z [11].

ideální bod (bod v nekonečnu). Pro $m = 0$ $\bar{\ell}$ leží na ose y'. Pro vertikální přímky ($m = \pm\infty$) bod $\bar{\ell}$ leží na ose x'.

Aby bylo možné reprezentovat čáry pro m v intervalu $(0, +\infty)$, přidá se další paralelní osa $-y'$. Tímto vzniká prostor x', $-y'$, který je identický s x', y' až na to, že osa y je invertovaná. Prostor x', y' se nazývá *straight* (značí se S) a prostor x', $-y'$ se nazývá *twisted* (značí se T). Kombinací obou prostorů vzniká prostor TS. Každá přímka je nyní reprezentována jako bod v *straight* $\bar{\ell_S}$ nebo *twisted* $\bar{\ell_T}$ polovině prostoru paralelních souřadnic

$$\begin{aligned} \bar{\ell_S} &= (d, b, 1-m)_{P^2}, \ -\infty \leq m \leq 0, \\ \bar{\ell_T} &= (-d, -b, 1+m)_{P^2}, \ 0 \leq m \leq \infty. \end{aligned} \tag{3.14}$$

Prostor TS je ukázán na obrázku 3.10, kde vlevo jsou zobrazeny tři body A, B a C a tři čáry ℓ_1, ℓ_2 a ℓ_3 procházející body. Střed prostoru $x - y$ je dán do středu obrázku z důvodu pohodlí. Vpravo je odpovídající $u - v$ prostor. Z tohoto obrázku si také můžeme všimnout, že konečný $u - v$ prostor je dostatečný:

$$\begin{aligned} -d &\leq u \leq d, \\ -\max\left(\frac{W}{2}, \frac{H}{2}\right) &\leq v \leq \max\left(\frac{W}{2}, \frac{H}{2}\right), \end{aligned} \tag{3.15}$$

kde W a H představují šířku a výšku vstupního snímku a d představuje vzdálenost os x' a y'. Z toho plyne, že každá přímka ℓ má právě jednu reprezentaci bodem $\bar{\ell}$ v prostoru TS. Toto neplatí v případech, kdy $m = 0$ a $m = \pm\infty$. Body obou těchto případů leží v obou prostorech na osách x' a y'. Díky tomuto můžou být prostory T a S spojeny jeden s druhým. Spojením os y' a $-y'$ vzniká Möbiova páska.

Detekce paralelních čar

Protože je čára v prostoru paralelních souřadnic reprezentována jedním bodem v originálním Eukleidovském prostoru, obrazy čar $\bar{\ell_1}, \ldots, \bar{\ell_n}$ leží na jedné čáře pouze tehdy, když $\ell_1, \ldots, \ell_n$ se protínají v jednom bodě. Když $\bar{\ell_1}, \ldots, \bar{\ell_n}$ sdílejí stejnou u souřadnici (jejich obrazy leží na vertikální čáře v prostoru paralelních souřadnic), bod protnutí je ideální bod (to znamená, že čáry jsou vzájemně paralelní). K ověření hypotézy kolinearity nalezených maxim se využívá RANSAC. Pro detekci čar shodných s jedním bodem, jehož reprezentace leží v prostorech S a T, existuje lineární transformace z S do T a zpět:

$$\begin{aligned} \forall \ell : \bar{\ell_S} = (u, v, 1)^2_P &\Rightarrow \bar{\ell_T} = \left(u, v, \frac{d - 2u}{d}\right)_{P^2}, \\ \forall \ell : \bar{\ell_T} = (u, v, 1)^2_P &\Rightarrow \bar{\ell_S} = \left(u, v, \frac{d + 2u}{d}\right)_{P^2}. \end{aligned} \tag{3.16}$$

3.3 Detekce figurek

Konvoluční neuronové sítě se v poslední době ukázaly jako dominantní a rychle se rozvíjející přístup v detekci a klasifikaci objektů ve snímcích. Jeden z hlavních důvodů je ten, že výsledky překonávají předešlé přístupy. Tato podkapitola se zabývá tématikou konvolučních neuronových sítí a popisuje prací použité modely a jejich rozdíly.

Konvoluční neuronové sítě

Konvoluční neuronová sít [18] (zkratka CNN z angl. *convolutional neural network*) je speciální typ umělé neuronové sítě [19], která přijímá na vstup snímky. CNN poskytují více škálovatelný přístup ke klasifikaci a rozpoznávání objektů ve snímku než předchozí přístup extrakce příznaků. Využívají principů lineární algebry, konkrétně násobení matic, k identifikaci vzorů v obraze. Z tohoto plyne, že CNN mohou být výpočetně náročné a vyžadující grafické procesní jednotky k trénování modelů.

Skládají se z vrstev uzlů. Tyto vrstvy se dají rozdělit na vstupní vrstvu, jednu nebo více skrytých vrstev a výstupní vrstvu. Typy vrstev v CNN jsou vrstvy konvoluční, sdružovací a plně propojené. Konvoluční vrstva je první vrstva konvoluční sítě. Následovat můžou další konvoluční vrstvy nebo sdružovací vrstvy. Plně propojená vrstva je finální vrstvou.

S každou vrstvou se zvětšuje komplexita CNN, identifikující větší části obrazu. Prvotní vrstvy se zaměřují na jednoduché příznaky, jako jsou barvy a hrany. Postupně, jak obraz prochází vrstvami, CNN začíná rozpoznávat větší obrazové elementy nebo tvary objektů, dokud nedojde k identifikaci objektu.

You only look once

Model YOLO (z angl. *You Only Look Once*) se řadí do jednostupňových regresivních konvolučních sítí [36]. To znamená, že v jednom průchodu obrazu neuronovou sítí dochází k predikci třídy objektu a jeho lokalizaci jako regresivní problém. Z tohoto důvodu má YOLO menší přesnost než dvoustupňové modely CNN, ale dochází ke kompenzaci zvýšením rychlosti detekce. Model YOLO získává na vstup celý obraz, což umožňuje naučit neuronovou síť i kontext objektu (v jakém prostředí se objekt vyskytuje).

YOLO rozdělí vstupní obraz do mřížky $S \times S$. Pokud se střed objektu nachází v buňce mřížky, tato buňka je zodpovědná za detekci tohoto objektu. Každá buňka predikuje B ohraničujících boxů a skóre důvěry pro tyto boxy. Tato skóre odrážejí, jak jistý si je model, že rámeček obsahuje objekt a jak přesně si myslí, že rámeček sedí objektu. Pokud se nevyskytuje žádný objekt v buňce, skóre by mělo být nula. Pokud se objekt vyskytuje v buňce, skóre se rovná IoU (z angl. *intersection over union*). Každý ohraničující rámeček se skládá z pěti hodnot: *[x, y, w, h]* a skóre, kde x a y představují souřadnice středu rámečku, w šířku, h výšku a skóre IoU mezi predikovaným rámečkem a reálným rámečkem objektu. Každý rámeček zároveň předpovídá C pravděpodobnosti výskytu všech tříd. Následně se spojí skóre pro ohraničující rámeček B a skóre předpovědi třídy C. Výsledek je mapa pravděpodobností výskytu specifického objektu. Posledním krokem je potlačení nemaximálních hodnot použitím prahu. Výsledné detekce jsou kódovány jako tenzor $S \times S \times (B \times 5 + C)$, kde $S \times S$ jsou rozměry mřížky, do kterých je obraz rozdělen, B je počet detekovaných rámečků na jednu mřížku a C představuje počet detekovaných tříd.

YOLO využívá vlastního druhu anotací pro jednotlivé ohraničující boxy. Tento formát je definován následovně *[cl, x, y, w, h]*, kde *cl* představuje třídu objektu, *(x, y)* souřadnice středu rámečku, w šířku a h výšku ohraničujícího boxu. Zároveň jsou tyto notace relativní k velikosti snímku. Souřadnice středu, šířka a výška rámečku jsou normalizovány rozměry obrázku. Výsledné hodnoty jsou reálná čísla v intervalu od 0 do 1.

Model YOLO je limitován počtem detekovaných rámečků, protože každá buňka mřížky předvídá jen 2 boxy a může obsahovat jen jednu třídu. Tímto je také limitován počet objektů, které mohou být blízko sebe detekovány. Obzvlášť tento problém vzniká při detekci malých objektů, které se nacházejí blízko sebe. Dalším limitujícím faktorem je problém s lokalizací. YOLO vnímá chyby malých a velkých ohraničujících rámečků jako sobě rovné. U velkých rámečků to nemusí mít takový vliv, ale u rámečků malých může dojít k velkému ovlivnění přesnosti lokalizace.

YOLOv3

Architektura modelu YOLOv3 je složená ze 106 vrstev a obsahuje 53 konvolučních vrstev [37]. Podle toho byla i pojmenována – *Darknet-53*. Oproti předchozí verzi (YOLOv2) se počet vrstev CNN zvětšil téměř třikrát [25]. Hlavním rysem YOLOv3 je to, že predikuje objekty ve třech měřítkách. Tato měřítka jsou přesně dána převzorkováním (*downsaplingem*) vstupního obrazu. To má za následek, že dochází k detekci desetinásobků rámečků oproti předchozí verzi. Z toho plyne, že YOLOv3 zaměnilo rychlost detekce za zvětšení přesnosti. Dalším rozdílem oproti předchozí verzi je zahrnutí tzv. *skip connections* a *upsamplingu*. Detekce v různých vrstvách pomáhá řešit problém s detekcí malých objektů. V předchozích verzích se využívala funkce *Softmax* k predikci tříd. To mělo za následek, že z každé buňky mřížky byl vybrán právě jeden objekt s nejvyšším skóre. YOLOv3 využívá nezávislých logistických klasifikátorů. To má za následek, že je nyní možné mít více tříd pro stejný objekt.

Obrázek 3.11: Ukázka vytvořených mozaiek YOLOv4. Obrázek je převzatý z [2].

YOLOv4

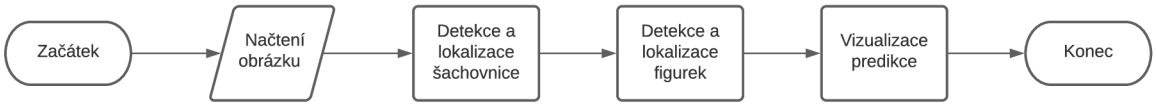
Joseph Redmon byl hlavním vývojářem YOLO. Svou práci ukončil z obav o zneužití jeho technologie v aplikacích pro armádu a problému ochrany soukromí [34]. Na jeho místo nastoupili vývojáři Alexey Bochkovskiy, Chien-Yao Wang a Hong-Yuan Mark Liao.

Architektura modelu YOLOv4 využívá páteřní sítě CSPDarknet53 [2]. Páteřní síť může zlepšit schopnost učení sítě. Byl přidán blok sdružovacích prostorových pyramid (z angl. *spatial pyramid pooling*). Tento blok zvětšuje receptivní pole a zlepšuje oddělování významných kontextových prvků. Využívá se síť *PANet* pro agregaci cest (z angl. *Path Aggregation Network*) jako metoda pro agregaci parametrů pro různé úrovně detektoru namísto *FPN* (z angl. *Feature Pyramid Networks*), která se používá v YOLOv3. Hlavní novinkou je mozaiková augmentace dat viz. 3.11. Při trénování dochází ke spojení čtyř obrázků dohromady. Další změnou je aktivační funkce. Předchozí model využívá aktivační funkci $ReLU$, zatím co YOLOv4 využívá funkci *Mish*. Také se začalo využívat k výpočtu chyby $CIoU$ (z angl. *Complete-IoU*) namísto IoU.

Kapitola 4

Lokalizace figurek na základě detekce šachovnice

Samotné řešení se dá rozdělit do tří hlavních modulů. Běh progmu je zobrazen v diagramu 4.1. Modul detekce a lokalizace šachovnice rozpoznává čáry v obraze, rozdělí je do dvou skupin ortogonálních čar, zvolí ideální čáry vůči středu šachovnice, vypočítá jejich protnutí a vrátí mřížku šachovnice. Modul detekce a lokalizace figurek rozpoznává figurky v obraze pomocí konvoluční neuronové sítě, potlačí nemaximální hodnoty, přiřadí figurky na políčka šachovnice a vrací reprezentaci šachovnice v podobě řetězce FEN. Poslední modul vytváří vizuální reprezentaci predikovaného stavu šachovnice z řetězce FEN.



Obrázek 4.1: Diagram postupu programu.

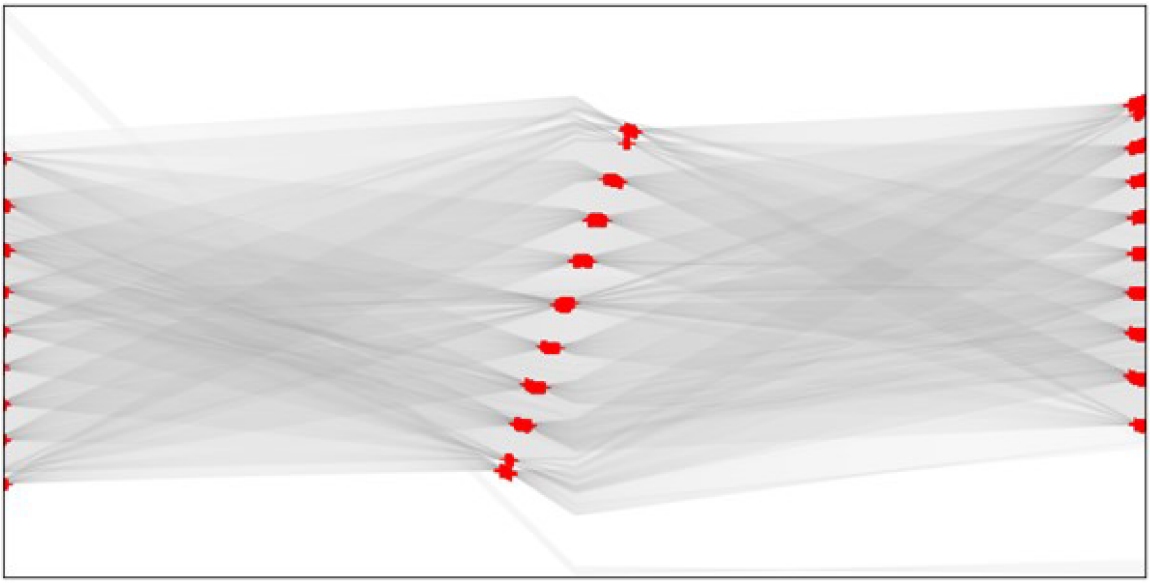
4.1 Detekce a lokalizace šachovnice

První návrh chtěl umožnit detekci šachovnice bez zásahu uživatele, ale vzhledem k negativním výsledkům lokalizace šachovnice to nebylo možné. Existující řešení využívají omezení, jako jsou například jasné barvy, prosté pozadí anebo nutnost označení celé šachovnice uživatelem. Na základě této vědomosti jsem se rozhodl, že práce bude využívat interakce s uživatelem k lokalizaci středu šachovnice. Tento přístup není tak omezující pro prostředí a zároveň není moc náročný na použití. Bylo zvážen i použití přístupu detekce rohů v obraze, ale vzhledem k detekci velkého množství nerelevantních rohů k šachovnici, detekce rohů figurek a omezení úhlu pohledu na šachovnici, jsem se rozhodl využít uživatelské interakce.

K detekci šachovnice byly navrženy dva programy. Jeden využívá paralelních souřadnic a druhý Houghovy transformace. Obě metody jsou založeny na detekci přímek a vyžadují předzpracování obrazu pomocí hranového detektoru. Použitý hranový detektor je Canny. Předpoklad pro detekci je pohled z pozice bílého nebo černého hráče.

Detekce pomocí paralelních souřadnic

Tento program využívá algoritmu založeném na Houghově transformaci, který využívá paralelních souřadnic k detekci čar [11]. Algoritmus v Pythonu byl implementovaný Romanem



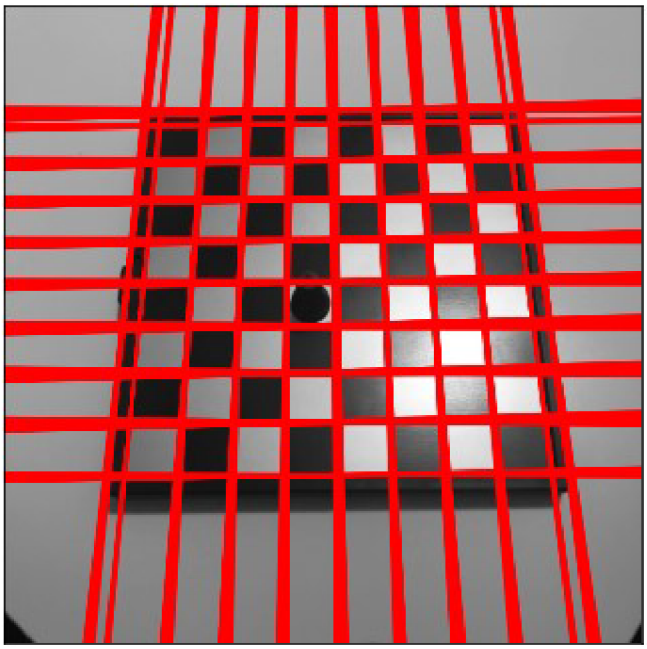
Obrázek 4.2: Naplněný akumulátor s zvýrazněnými maximy.

Juránkem [21]. K implementaci byly využity knihovny *OpenCV*, *numpy*, *math* a *statistics*. Modul je implementován v souboru *chessboardDetectionPClines.py* ve třídě *ChessBoardDetection*. Běh detekce se spouští funkcí *detect_chessboard*, která přijímá parametr *image*, který představuje cestu ke vstupnímu snímek. Výstupem programu je mřížka středů políček.

Běh programu začíná načtením černobílého vstupního snímku a změnou jeho velikosti do rozměrů 600×600. Použije se nelineárního bilaterálního filtru implementovaného v knihovně OpenCV k vyhlazení snímku a tím redukci šumu. Filtr je implementován ve funkci *bilateralFilter* v knihovně OpenCV. Výhodou zmíněného filtru je, že zachovává hrany. Po předzpracování vstupního obrazu se detekují hrany pomocí Cannyho hranového detektoru, který je popsán v kapitole 3.2. Ze zpracovaného obrazu se načtou lokace hran použitím prahu intenzity a také jejich váhy. Vytvoří se akumulátor paralelních souřadnic a jsou do něj načteny lokace a váhy hran. V akumulátoru se vyhledají lokální maxima viz. obrázek 4.2 a je provedena jejich zpětná transformace na parametry (a, b, c) z obecné rovnici přímky $ax + by + c = 0$. Výsledné přímky ℓ jsou reprezentovány ve dvou seznamech souřadnic X, Y, které byly vypočítány protnutím přímek s hranami snímku. Přímky jsou zobrazené na obrázku 4.3. Jejich reprezentace je následovná

$$\begin{aligned} X_i &= [x_1, x_2], \\ Y_i &= [y_1, y_2], \\ \ell_i &= [x_1, y_1, x_2, y_2]. \end{aligned} \tag{4.1}$$

Detekované přímky jsou uloženy v jednom listu bez ohledu na jejich vztah. Z tohoto důvodu jsou dále přímky rozděleny na horizontální a vertikální. Tento krok se provádí ve funkci *split_lines*. Vzhledem k tomu, že detekované čáry jsou uloženy jako nekonečné přímky, rozdělení probíhá na základě rozdílů x a y hodnot. V případě, že je přímka horizontální, rozdíl hodnot x je velikost obrázku a hodnoty y jsou podobné. Pro vertikální přímky to funguje obdobně, ale bere se v potaz rozdíl hodnot y a jsou porovnávané x hodnoty. Zároveň si v obrázku 4.3 můžeme všimnout, že dochází k detekci více čar na jednu hranu. Tento problém

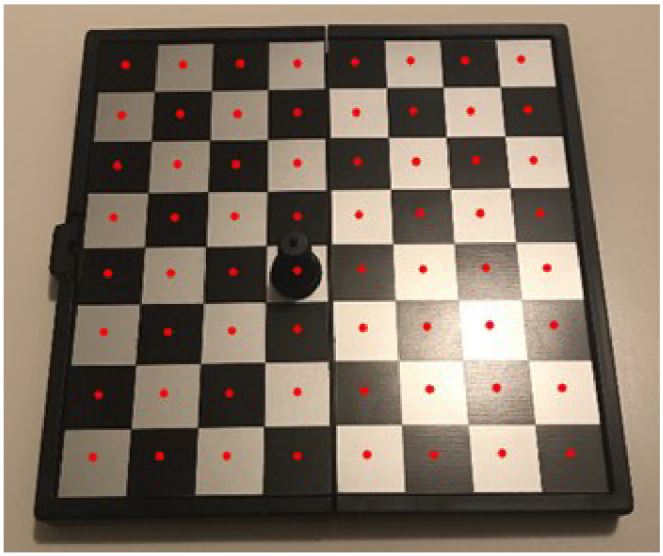


Obrázek 4.3: Detekované čáry odpovídající maximům akumulátoru.

se řeší spolu s rozdělováním přímek a to kontrolou, zda se podobná přímka již vyskytuje v listu vybraných přímek. Pro vertikální přímky se porovnávají hodnoty x a pro horizontální hodnoty y.

Po detekci a rozdělení přímek je potřeba zkontrolovat, zda došlo k nalezení minimálního počtu horizontálních a vertikálních přímek. Minimální počet přímek k přesnému určení šachovnice je 7 a 7. Následně je potřeba, aby uživatel zadal střed šachovnice. Pro tento krok je vytvořeno okno 600×600 s původním snímkem. Uživatel zadá střed šachovnice kliknutím do obrázku. Kliknutí zavolá funkci *set_middle_point* k obsluze události, která uloží souřadnice kliknutí a umožní programu pokračovat ve vykonávání.

Pomocí informaci o středu šachovnice dojde následně ke zvolení vhodných vertikálních a horizontálních přímek ve funkci *filter_lines*. Tento proces začíná nalezením přímky nejbližší ke středu a rozdělením přímek. Horizontální přímky se rozdělí na horní a dolní a vertikální přímky se rozdělí na levé a pravé vůči přímce, která je nejblíže ke středu. Toto rozdělení je nutné vzhledem k horizontálním přímkám. Vzdálenost mezi jednotlivými přímkami v horní části obrazu se liší od vzdáleností v dolní části (vzdálenosti přímek v horní části šachovnice jsou menší než vzdálenosti v dolní části, což je způsobeno úhlem pohledu). Toto by se dalo vyřešit zvětšením přijatelné odchylky, ale toto řešení není ideální, protože může dojít ke zvolení chybně detekovaných přímek. Dále dojde k výpočtu vzdáleností jednotlivých přímek a výpočtu mediánu vzdáleností. Podle mediánu je určeno, zda následující přímka je vhodná nebo ne. Pokud je přímka vhodná, dojde k jejímu uložení a aktualizování hodnoty vzdálenosti pro hledání další přímky. Tímto může zůstat odchylka nízká a nedochází k vybírání špatných přímek. Tento přístup řeší problém s horizontálními přímkami, ale je náchylnější k problému s velkým množstvím chybných detekcí, což by mohlo mít za následek nízkou prvotní vzdálenost, a tudíž nenalezení ideálních přímek. Počet volených přímek je omezen na osmnáct (devět pro horizontální a devět pro vertikální z toho pět pro každou polovinu, kde počáteční přímka je shodná pro obě poloviny).



Obrázek 4.4: Vizualizace středů políček.

Po vybrání vhodných přímek dojde znovu ke kontrole, zda jich je stále alespoň minimální potřebný počet k lokalizaci šachovnice. Pokud je více přímek jednoho druhu než druhého nebo je přímek více jak devět, dojde k vyrovnání počtu a odstranění přebytečných přímek ve funkci *remove_excess_lines*. Tento krok se vykoná, protože šachovnice je jasně identifikována devíti přímkami a k nim devíti ortogonálními přímkami.

Výsledkem tohoto modulu jsou středy jednotlivých políček šachovnice. Předtím, než jsme schopni vypočítat jejich středy, potřebujeme vypočítat pozice všech rohů políček. Rohy políček získáme vypočtením protnutí přímek ve funkci *calculate_lines_intersects*. Pokud je počet přímek menší jak osmnáct, je potřeba dopočítat postranní body. Tento krok řeší funkce *calculate_corners*, která využívá parametrického vyjádření přímky. Vypočítá sklon a protnutí s osou y přímky a bod je posunut do strany o medián vzdáleností. Po získání všech rohů políček funkce *calculate_center_positions* vypočítá jejich středy průměrováním vzdáleností rohů políčka.

Výsledné středy políček je možné zobrazit funkcí *display_center_points*, která promítne vypočítané body do původního snímku viz. obrázek 4.4.

Detekce Hoghovou transformací

Hlavní princip kroků tohoto programu je stejný jako u detekce založené na PClines. Rozdíly se vyskytují v metodě detekce čar a zpracování přímek. Použitá metoda k detekci čar je založená Houghova transformace. Použité knihovny jsou OpenCV, numpy, math a statistics. Modul je implementován v souboru *chessboardDetectionHoughlines.py* ve třídě *ChessBoardDetection*. Spuštění detekce se provádí zavoláním funkce *detect_chessboard*, jejíž parametrem je cesta ke snímku. Stejně jako modul založený na PClines, výstupem je mřížka středů políček šachovnice.

Počátkem běhu, stejně jako v PClines, je načtení černobílého vstupního snímku a jeho předzpracování obrazu pro detekci čar. Využije se bilaterální filtr a detekují se hrany pomocí Cannyho detektoru hran. Na takto zpracovaný obraz se spustí metoda *HoughLinesP*, která je implementována v knihovně OpenCV. Funkce vrací čtyř-prvkové vektory (x_1, y_1, x_2, y_2)

reprezentující čáry, kde (x_1, y_1) označuje počátek a (x_2, y_2) označuje konec detekovaného segmentu čáry.

Stejně jako v modulu PClines jsou čáry uloženy v jednom seznamu a je potřeba je rozdělit na horizontální a vertikální. Na rozdíl od algoritmu PClines, funkce HoughLinesP vrací jen detekované segmenty. To znamená, že nedetekuje nekonečné přímky, ale jen úsečky. Z tohoto důvodu se nemůže využít stejného přístupu. Funkce *split_lines* vypočítá směr přímek pomocí rovnice $m = \frac{\Delta y}{\Delta x}$ a normalizuje výsledek. Tím vzniknou vektory směrů pro obě osy. Přímky jsou následně roztříděné podle těchto hodnot. Pokud se hodnota x vektoru blíží k 1, jedná se o horizontální přímku a pokud se hodnota vektoru y blíží k 1, jedná se o vertikální přímku. Zároveň se při tomto rozřazování kontroluje, zda již je uložena podobná přímka, aby nedocházelo k uložení stejných nebo hodně podobných přímek.

Vzhledem k tomu, že na jedné hraně může být detekováno více čar různých velikostí, je potřeba podobné přímky sjednotit. Funkce *find_similar_lines* porovnává souřadnice čar v předem určeném rozmezí a shlukuje podobné čáry. Funkce *check_similar_clusters* zkontroluje shluky podobných čar, zda neexsitují shluky, které již jsou obsaženy v jiném, větším shluku čar. Jednotlivé shluky čar jsou následně zprůměrovány a je vytvořena jedna přímka pro daný shluk ve funkci *join_lines*.

Dále je zkontrolováno, zda byl nalezen minimální počet čar potřebných pro identifikaci a, stejně jako v modulu PClines, je uživatel požádán o zadání středu šachovnice.

Před výpočtem protnutí nalezených čar je potřeba čáry prodloužit do nekonečna, aby byla nalezena všechna jejich protnutí. K prodloužení čar je potřeba vypočítat jejich sklon a protnutí s osou y. Sklon se vypočítá ve funkci *calculate_slopes* a protnutí s osou y se vypočítá ve funkci *calculate_y_intersect*. Následně je možné prodloužit čáry. Využívá se parametrického vyjádření přímky $y = mx + b$. Funkce *extend_horizontal_lines* prodlužuje horizontální čáry podle následujících rovnic

$$\begin{aligned} x_1 = 0,\ x_2 = w, \\ y_i = m \cdot x_i + b, \end{aligned} \tag{4.2}$$

kde m je sklon, b protnutí s osou y a w je šířka snímku. Funkce *extend_vertical_lines* prodlužuje vertikální čáry podle rovnic

$$\begin{aligned} y_1 = 0,\ y_2 = h, \\ x_i = \frac{(y_i - b)}{m}, \end{aligned} \tag{4.3}$$

kde h je výška snímku. Dalším potřebným krokem před výpočtem protnutí přímek je uspořádat přímky. Vertikální přímky jsou uspořádány zleva doprava a horizontální shora dolů. Následně jsou vybrány ideální vertikální a horizontální přímky. Zbytek průběhu programu je identický s modulem PClines. Z tohoto důvodu nebude znovu zmiňován.

4.2 Detekce a lokalizace figurek

Podkapitola popisuje přípravu, spuštění a průběh trénování konvolučních neuronových sítí. Byly natrénovány dva modely YOLOv3 a YOLOv4. K trénování byl využit framework *Darknet*. Trénování probíhalo na Google Colab a zdrojový soubor *yolo.ipynb* je dostupný na médiu. Modul je implementován v souboru *figureDetection.py* pomocí knihovny OpenCV. Detekce se spouští funkcí *detect_figures*, jejíž parametry jsou cesta ke snímku a mřížka detekované šachovnice.

Program na detekci a lokalizaci figurek prvně načte vstupní snímek, konfigurační soubor konvoluční neuronové sítě a její váhy. Oba modely provádějí detekci na obrázku o velikosti 416×416. Do sítě se vloží obrázek a zachytí se výstup CNN. Následně se prochází všechny detekce ve výstupu a všechny detekce, které přesahují hranici důvěry, jsou uloženy. YOLO má vlastní konvenci ukládání ohraničujících boxů. Ukládá si souřadnice (x, y) středu rámečku, jeho výšku a šířku. Následně dojde k potlačení ne-maxim. To znamená, že pokud se někde překrývají boxy, zvolí se ten s větší důvěrou. Posledním krokem je nalezení políčka, kam figurka patří. K tomuto se využívá výstup předchozího modulu detekce šachovnice. Výstupem zmíněného modulu je mřížka středů políček. Postup přiřazování figurky na políčko se dá přirovnat tzv. *„brute force“* algoritmu. Ke každému bodu v mřížce šachovnice je spočítána vzdálenost a na nejmenší z nich se figurka přiřadí. Výstupem tohoto modulu je řetězec FEN, který představuje predikci aktuálního stavu šachovnice.

Trénování modelů neuronových sítí

K učení byl využit framework Darknet [35], který je implementován v programovacích jazycích C a CUDA. Darknet podporuje výpočty na grafickém procesoru, což urychluje celý proces učení. Trénování neuronových sítí probíhalo na *Google Colab*. Byly použity předem trénované váhy jednotlivých modelů, protože trénování od začátku je časově náročné.

Před samotným trénováním je nutné připravit potřebné soubory pro trénování a zkompilovat framework Darknet. Před kompilací je potřeba upravit Makefile, aby bylo možné trénovat na grafickém procesoru s použitím knihovny OpenCV. Konkrétně se povolí použití knihoven CUDA, cuDNN a OpenCV. Knihovna CUDA umožňuje trénování na grafickém procesoru, cuDNN akceleruje výpočty a OpenCV zpracovává obrázky. Dále je potřeba upravit konfigurační soubory architektur neuronových sítí. V případě modelu YOLOv3 se jedná o soubor *yolov3.cfg* a pro model YOLOv4 se jedná o soubor *yolov4.cfg*. Tyto soubory zároveň obsahují nastavení sítí.
Parametry nastavení sítě byly změněny následovně:

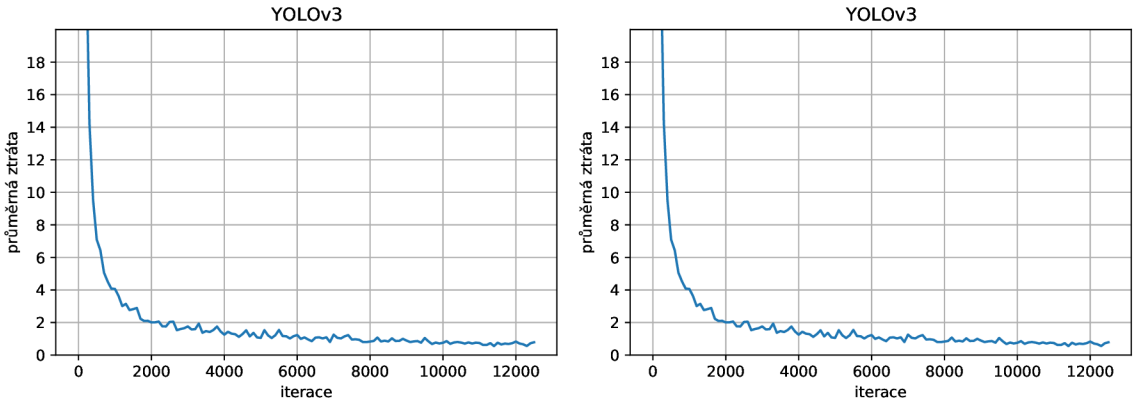
- **Dávka** - 64 pro trénování, 1 pro testování.
- **Členění** - 16 pro trénování, 1 pro testování.
- **Šířka a výška snímku** - 416×416.
- **Maximální počet iterací** - 24000.
- **Kroky** - 80%, 90% z maximálního počtu iterací.

Také je potřeba upravit určité nastavení vrstev sítě. Pro model YOLOv3 následující:

- **610, 696, 783** - *classes=12* (počet objektových tříd)
- **603, 689, 776** - *filters=51* (počet objektových tříd +5) $\times$ 3

Pro model YOLOv4:

- **970, 1058, 1146** - *classes=12* (počet objektových tříd)
- **963, 1051, 1139** - *filters=51* (počet objektových tříd +5) $\times$ 3



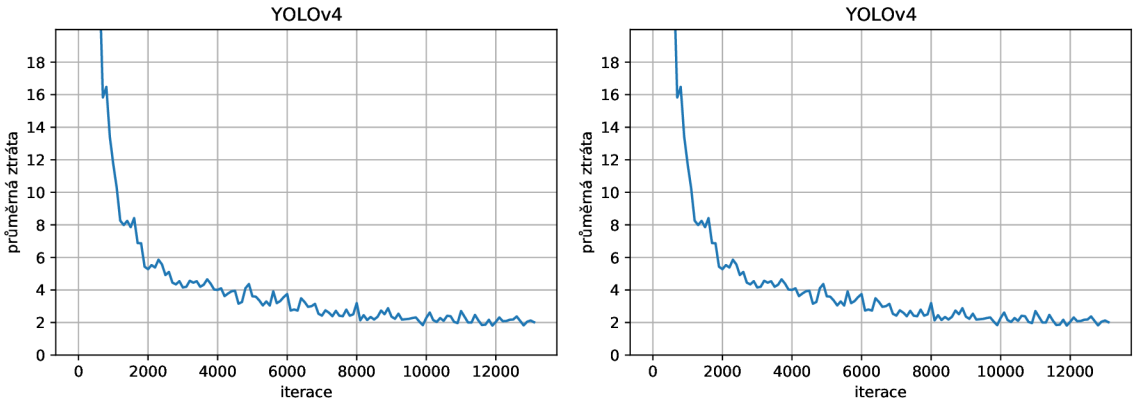
Obrázek 4.5: Graf průměrné ztráty (vlevo) a průměrné přesnosti (vpravo) modelu YOLOv3.

Posledním krokem před trénováním je vytvoření souborů *obj.names*, *obj.data*, *train.txt* a *valid.txt*. Soubor *obj.names* obsahuje názvy tříd objektů, kdy jejich pozice odpovídá jejich identifikátoru. *Obj.data* obsahuje inforamce pro trénovací program (počet tříd a cesty k souborům *obj.names*, *train.txt*, *valid.txt* a lokaci, kam se budou ukládat zálohy). Soubory *train.txt* a *valid.txt* obsahují cesty k obrázkům dané datové sady. K jejich vytvoření byl napsán skript, který se nachází v souboru *yolo.ipynb*.

Trénování probíhalo na Google Colab, kde byl vytvořen soubor *yolo.ipynb* obsahující všechno potřebné pro replikaci tréninku modelů. Google Colab, také znám jako *Google Colaboratory*, je volně dostupná webová aplikace založená na *Jupyter notebook*, který běží na platformě *Google Cloud*. Kromě toho jsou tam nainstalované knihovny pro manipulaci s daty a trénování modelů strojového učení. Hlavní výhodou je možnost použití grafického procesoru, dostupnost a jednoduché sdílení. Dostupné GPU jsou Nvidia K80, T4, P4 a P100. Nevýhodou je omezený procesorový čas a občasné výpadky, které mohou zapříčinit ztrátu všech dat.

Darknet sám od sebe neukládá záznam událostí při trénování do souboru a vzhledem k tomu, že občas dochází k výpadkům nebo terminaci procesu ve službě Google Colab, bylo potřeba upravit trénovací soubor *detector.c* frameworku Darknet k tomu, aby zapisoval data o iteracích, průměrné ztrátě a přesnosti do zvoleného souboru. Díky těmto informacím je možné vytvořit grafy průměrné ztráty a průměrné přesnosti. K tomuto jsem vytvořil jednoduchý skript v Pythonu *plotAvgLoss.py*, který vykreslí grafy průměrné ztráty a přesnosti. Průměrná ztráta představuje jak špatně nebo dobře se model chová po každé iteraci optimalizace. Metrika přesnosti se používá k měření výkonu interpretovatelným způsobem. Jedná se o měřítko toho, jak přesná je předpověď ve srovnání se skutečnými daty. Tyto grafy jsou zobrazeny v obrázcích 4.5 a 4.6.

Výstupem trénování jsou soubory obsahující váhy. K ukládání těchto souborů dochází pravidelně po určitém počtu iterací (nejčastěji 100). Díky průběžnému ukládání vah, se při výpadku může ztratit jen malý pokrok v učení. Vždy se ukládá aktuální nejlepší výsledek na základě přesnosti (MAP.05) a aktuální stav. Další výhodou průběžného ukládání je fakt, že při přeučení modelu na datové sadě můžeme vybrat ideální váhy.



Obrázek 4.6: Graf průměrné ztráty (vlevo) a průměrné přesnosti (vpravo) modelu YOLOv4.

Obrázek 4.7: Použité SVG obrázky figurek. Převzato z [20].

4.3 Vizualizace šachovnice

První návrh realizace dvourozměrného zobrazení detekované šachovnice byl pomocí vektorových obrázků. Podle výstupu modulu detekce a lokalizace figurek se na jednotlivá políčka zobrazily vektorové obrázky figurek. Toto řešení bylo dostačující k vizualizaci šachovnice, ale vzhledem k zájmu a možnému dalšímu vývoji práce se rozhodlo, že se použije knihovny *Chess* a jejího modulu *Chess.svg*. K zobrazení okna s vizualizací predikce stavu šachové partie se využívá knihovny *PyQT5*.

Knihovny jsou implementovány v Pythonu. Knihovna Chess zastává funkcí generace a validace jednotlivých kroků. Knihovna Chess.svg je modul knihovny Chess, který vykresluje šachovnici a SVG obrázky figurek. Podporovány jsou běžné formáty notací šachu (FEN) a různé herní typy.

Vizualizace získává od modulu detekce a lokalizace šachovnice řetězec FEN. Tento řetězec reprezentuje aktuální stav šachovnice. V našem případě představuje predikci stavu. Knihovna Chess si pomocí funkce *Board* z řetězce FEN vytvoří reprezentaci šachovnice. Tato reprezentace je následně předána modulu Chess.svg, který vytvoří SVG reprezentaci šachovnice. SVG reprezentace šachovnice je následně zobrazena uživateli viz. obrázek 4.8.

Obrázek 4.8: Ukázka procesu lokalizace figurek.

Kapitola 5

Datová sada

Kapitola popisuje výslednou datovou sadu, jakým způsobem byla pořízena a jak byla upravena pro trénování a testování modelů konvolučních neuronových sítí. Také obsahuje informace týkající se implementace skriptů pro automatické anotování dat a tvorby ohraničujících rámečků do formátu YOLO.

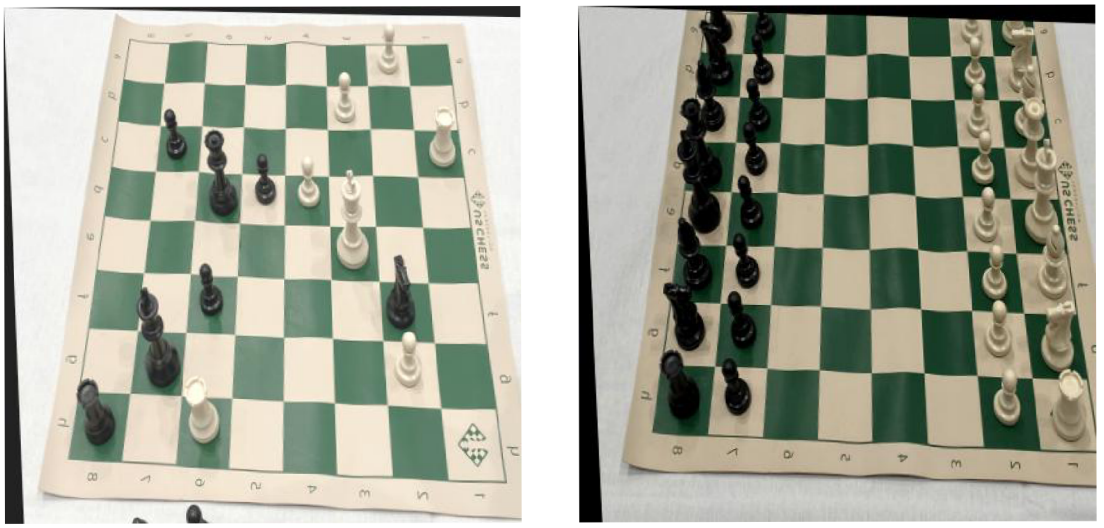
Přesnost detekčního modelu neuronové sítě je závislá na kvalitě datové sady. Pro nejlepší možný výsledek je důležité mít dostatečně velké množství různorodých obrázků. Vzhledem k tomu, že tato práce cílí k robustní detekci a klasifikaci figurek, byla snaha získat fotky různorodých šachových setů.

5.1 Získání dat

Ačkoliv se podobným problémům věnovalo již mnoho výzkumníků a fandů počítačového vidění, v raném stádiu průzkumu jsem zjistil, že neexistuje prakticky žádná volně dostupná datová sada šachových setů. Každé existující řešení si vytvářelo vlastní datovou sadu (většinou o jednom druhu šachovnice a šachových figurek), kterou následně ani nepublikovalo. Podařilo se mi nalézt jednu volně dostupnou datovou sadu. Tato datová sada byla publikována webovým portálem *Roboflow* [40]. Všechny fotky byly pořízeny pod stejným úhlem. Obsahuje 693 obrázků šachového setu a celkově obsahuje 7083 anotací, které jsou rozděleny v tabulce 5.1. Ukázka datové sady je na obrázku 5.1.

Jediným problémem, který vznikl při zahrnutí této datové sady, byl rozpor anotací figurek s předešlými získanými daty. Datová sada Roboflow má anotace tříd v jiném pořadí než mnou získaná datová sada viz. tabulka 5.2. Tento problém byl vyřešen jednoduchým skriptem v Pythonu, který prošel jednotlivé soubory anotací a přepsal původní třídy na požadované třídy.

Celkově bylo pořízeno více jak dva tisíce snímků deseti druhů šachových setů, ze kterých bylo vybráno 1709 snímků. Nevyhovující snímky byly odebrány z důvodu rotací šachovnice, nevyhovujících úhlů, nevyhovujících dat figurek (položené, otočené anebo jinak nevhodně situované) a nevyhovujících dat šachových setů (například skleněné šachové sety). Trénovací datová sada obsahuje celkově 1350 fotografií a 15878 anotací. Rozložení anotací na jednotlivé figurky je zobrazeno v tabulce 5.3. Validační datová sada obsahuje celkem 359 snímků a 4118 anotací. Rozložení anotací je zobrazeno v tabulce 5.4.



Obrázek 5.1: Ukázka datové sady Roboflow.

Figurka	**Počet anotací**
Bílý pěšák	1587
Bílý střelec	422
Bílý kůň	464
Bílá věž	460
Bílá královna	273
Bílý král	359
Černý pěšák	1651
Černý střelec	334
Černý kůň	478
Černá věž	495
Černá královna	211
Černý král	349

Tabulka 5.1: Počet anotací na figurku v datové sadě Roboflow [40].

Název figurky	**Třída Roboflow**	**Třída figurky**
Bílý král	0	5
Bílá královna	1	4
Bílý střelec	2	3
Bílý kůň	3	2
Bílá věž	4	1
Bílý pěšák	5	0
Černý král	6	11
Černá královna	7	10
Černý střelec	8	9
Černý kůň	9	8
Černá věž	10	7
Černý pěšák	11	6

Tabulka 5.2: Vizualizace problému inkonzistence klasifikátoru tříd.

Název figurky	**Počet anotací**
Bílý pěšák	3372
Bílý střelec	953
Bílý kůň	990
Bílá věž	1196
Bílá královna	722
Bílý král	822
Černý pěšák	3527
Černý střelec	845
Černý kůň	951
Černá věž	1107
Černá královna	578
Černý král	805

Tabulka 5.3: Počet anotací na figurku v trénovací datové sadě.

Název figurky	**Počet anotací**
Bílý pěšák	885
Bílý střelec	237
Bílý kůň	269
Bílá věž	321
Bílá královna	172
Bílý král	208
Černý pěšák	894
Černý střelec	236
Černý kůň	247
Černá věž	286
Černá královna	158
Černý král	205

Tabulka 5.4: Počet anotací na figurku ve validační datové sadě.

5.2 Anotace dat

Vzhledem k tomu, že většina dat v datové sadě byla získána mnou nebo mými kamarády, je potřeba data anotovat. Byly vytvořeny pomocné skripty pro anotaci pozice šachovnice a pozice šachových figurek. Dále byl vytvořen skript, který následně z těchto anotací vytvoří kvádry okolo políček s figurkami a z nich ohraničující boxy pro konvoluční neuronovou síť v notaci YOLO.

Anotace rohů šachovnice

Pomocný skript pro anotování šachovnice byl napsán v jazyce Python a implementován v souboru *boardAnotation.py* ve třídě *BoardAnnotation*. Běh se spouští zavoláním funkce *create_annotation*, která má parametr *directory* představující cestu k adresáři s obrázky. Byl vytvořen použitím knihovny OpenCV.

Prvním krokem programu je načítí obrázky ze složky. Následně se každý obrázek postupně otevře v okně 600×600 pro zadání rohů šachovnice. Šachovnice se anotuje počínaje v levém horním rohu po směru hodinových ručiček. Po kliknutí se objeví bod v místě kliknutí. Po anotaci všech čtyř rohů šachovnice se klávesou n uloží pozice bodů do souboru *název_souboru_board.txt* a přepne se na další obrázek. Obrázky, které již mají vytvořený soubor s anotací pozice šachovnice, jsou přeskočeny. Soubory s uloženými souřadnicemi jsou v následujícím formátu

$$X_{TL}, Y_{TL}$$
$$X_{TR}, Y_{TL}$$
$$X_{DR}, Y_{DR}$$
$$X_{DL}, Y_{DR}$$

kde TL označuje horní levý roh, TR horní pravý roh, DR dolní pravý roh a DL dolní levý roh.

Anotace figurek

Pomocný skript pro anotaci figurek byl napsán v jazyce Python a implementován v souboru *figureAnotation.py* ve třídě *FigureAnnotation*. Běh programu se spouští zavoláním funkce *create_annotation*, jejíž parametr *directory* představuje cestu do adresáře, který obsahuje obrázky. K vytvoření byla použita knihovna OpenCV.

Běh programu začíná načtením obrázků z adresáře. Po načtení všech obrázků se vytvoří dvě okna. První okno obsahuje pomocný destinační obrázek pro anotaci pozic a druhé okno obsahuje zdrojový snímek, který se anotuje. V této části práce se prvně využívalo anotací rohů šachovnice, ale tím byla vytvořena povinnost je prvně anotovat. V některých případech, jako například při zahrnutí jiných obrázků na testování detekce a lokalizace, je to zbytečná práce. Z tohoto důvodu se využívá pomocného destinačního obrázku. Program iterativně prochází skrze obrázky a uživatel je anotuje. Pokud se nejedná o první obrázek ze složky, bude zobrazen i předchozí obrázek. Předchozí obrázek je zobrazen pro umožnění rotace pozic nebo uložení stejných pozic (to se využije pro stejné figurkové rozestavení, které bylo otočeno anebo při změně pohledu na šachovnici). Pozice figurek se anotují kliknutím na jejich políčko v destinačním obrázku. Následně je potřeba v konzoli zadat typ figurky. Další možností anotace je také využít klávesy *f*, která umožní zadat název figurky a anotovat více políček najednou. Typ figurky se zadává zkratkou podle následující tabulky:

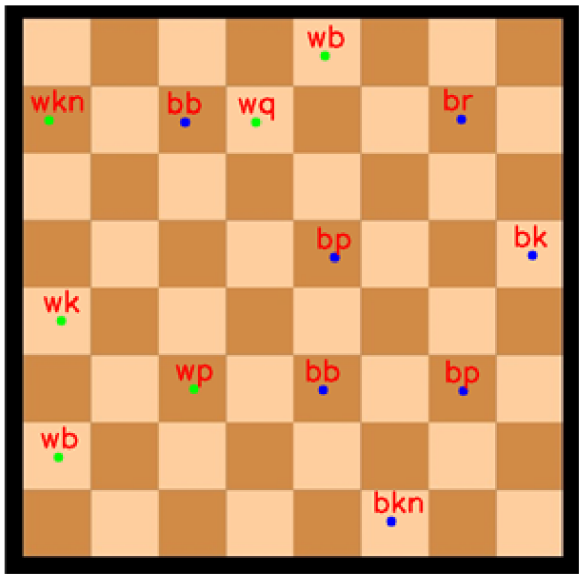
Obrázek 5.2: Ukázka anotace desky.

Název figurky	**Zkratka figurky**
Bílý pěšák	wp
Bílý střelec	wb
Bílý kůň	wkn
Bílá věž	wr
Bílá královna	wq
Bílý král	wk
Černý pěšák	bp
Černý střelec	bb
Černý kůň	bkn
Černá věž	br
Černá královna	bq
Černý král	bk

Tabulka 5.5: Tabulka zkratek pro jednotlivé figurky.

Zadáním figurky se zobrazí na obrázku zkratka figurky s odlišenou barvou v místě kliknutí. Po zadání všech figurek, tedy ukončení anotace konkrétního obrázku, se pomocí klávesy *n* zapíšou body do souboru *název_obrázku_pos.txt* a zobrazí se další obrázek. V případě, že je obrázek stejný jako předchozí nebo došlo jen k otočení šachovnice, se může klávesou *r* dostat do režimu přepočítání pozic z předešlých pozic. Zde jsou možnosti *rl*, *rr* a *s*, kde *rl* představuje otočení doleva, *rr* otočení doprava a *s* stejnou pozici. Otočení jsou realizována rovnicemi

$$\begin{aligned} rl &: x = y,\ y = 7 - x, \\ rr &: x = 7 - y,\ y = x. \end{aligned} \tag{5.1}$$



Obrázek 5.3: Anotace figurek obrázku 5.2.

Struktura ukládání figurek je *cl, x, y*, kde *cl* představuje identifikátor třídy, *x* a *y* souřadnice políčka na šachovnici (A-H, 1-8).

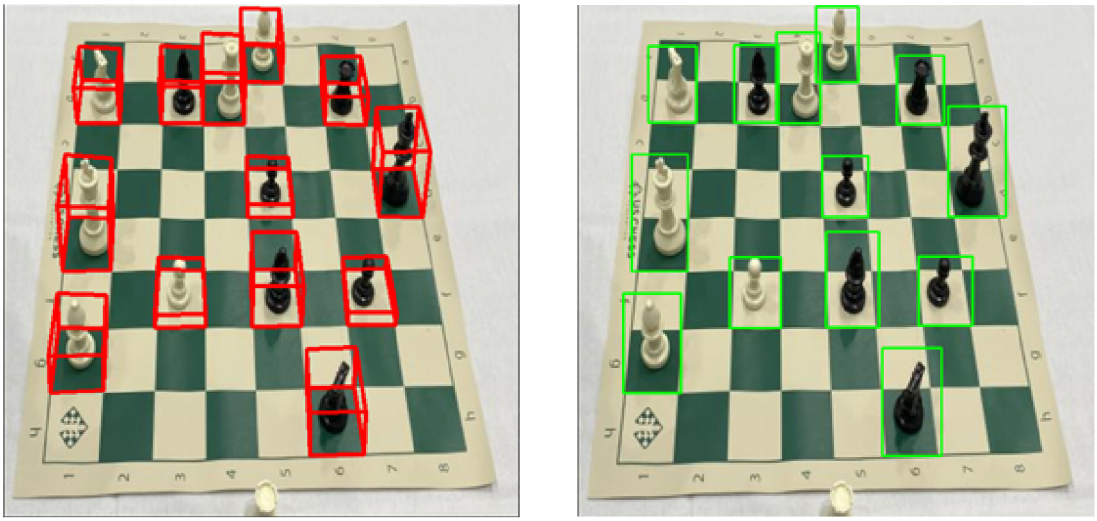
Tvorba ohraničujících rámečků

Vytváření anotací je samo o sobě časově náročné a obzvlášť ve velkém množství dat. Z tohoto důvodu se rozhodlo, že se tato činnost bude automatizovat. Toto je možné, protože šachovnice má jednotlivá políčka přesně lokalizovaná. Z toho plyne, že pokud se označí rohy šachovnice, dokážeme určit pozici jakéhokoliv políčka na šachovnici.

Program byl implementován pomocí Pythonu v souboru *boudingBoxes.py*. Využívá knihoven OpenCV a Numpy a anotací rohů šachovnice a pozic figurek. Tvorba ohraničujících boxů nevyžaduje použití vytvořených pomocných skriptů. Vyžaduje jen stejný formát souborů a jejich názvů.

Program je implementován ve třídě *BoundingBoxes*. Program je spuštěn zavoláním funkce *create_bounding_boxes*. Funkce má parametry *directory*, *draw_boxes* a *sizes*. Parametr *directory* představuje cestu ke složce s obrázky, *draw_boxes* je nepovinný parametr s booleovskou hodnotu k určení, zda uživatel chce vykreslovat kvádry políček a parametr *sizes* je nepovinný parametr specifikující velikosti jednotlivých figurek.

Vstupním parametrem programu je cesta ke složce, ze které se jednotlivé snímky načtou. Pokud jsou zadány specifické velikosti figurek, aktualizujeme výchozí velikosti. Následně se iterativně prochází skrze načtené obrázky a kontroluje se, zda existují soubory s lokací figurek a anotacemi rohů šachovnice. Pokud existují, program je načte. V případě, že soubory neexistují, podá se uživateli informace, že soubory neexistují pro specifický snímek a běh programu se zastaví. Vypočítá se homografie a projekční matice kamery. Homografie je vypočítána funkcí *findHomography* z načtených rohů šachovnice pro konkrétní snímek a z rohů v pomocném obrázku. V destinačním obrázku dokážeme přesně určit všechny rohy políček šachovnice. Tyto rohy pomocí homografie převedeme do skutečného snímku a z pozic těchto bodů dochází k výpočtu projekční matice kamery. Parametry kamery jsou



Obrázek 5.4: Vykreslené kvádry (vlevo) a ohraničující rámečky (vpravo).

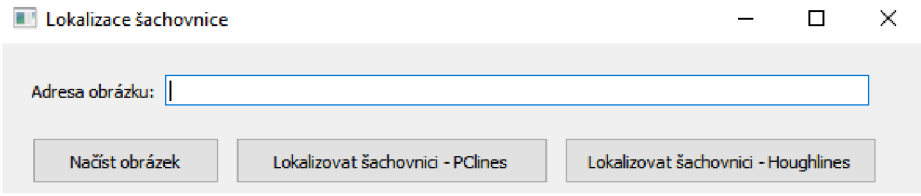
vypočítány funkcí *calibrateCamera*. K vykreslení kvádru okolo políčka je nejprve potřeba odhadnout pozici kamery. Funkce *solvePnP*, která využívá předem zjištěné projekční matice kamery a rohů políčka, na kterém se figurka nachází, vypočítá pozici kamery v obrazové rovině. Pozice figurky se získá pomocí homografie z destinačního obrázku. Dále se promítnou 3D body funkcí *projectPoints* do obrazové roviny a získáme kvádr okolo políčka šachovnice. Program zároveň disponuje funkcí *display_bounding_rectangles*, která umožňuje vykreslení samotných ohraničujících rámečků v YOLO konvenci viz. 5.4 vpravo.

Kapitola 6

Demo aplikace

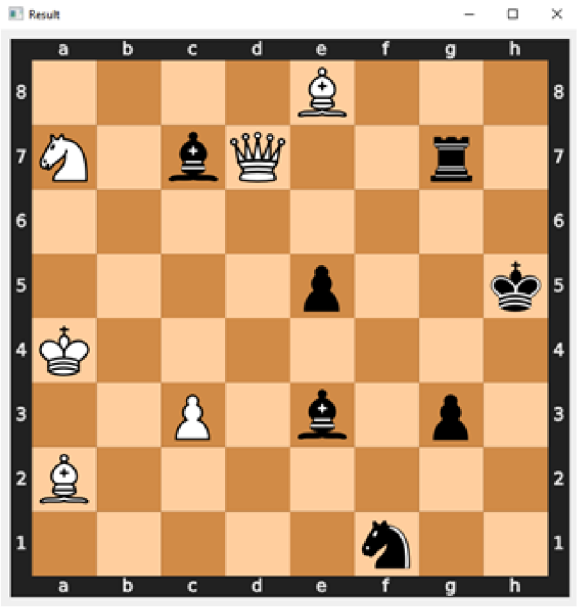
Vzhledem k tomu, že tato práce byla zaměřená na detekci a lokalizaci šachovnice, jevilo se ideálním vytvořit jednoduchou demo aplikaci, která ukáže funkčnost. Aplikace byla implementována v jazyce Python použitím knihovny *PyQt5* v souboru *demo.py*. Je založena na principu návrhu KISS („keep it simple stupid“). Demo aplikace má jedno hlavní okno zobrazené na obrázku 6.1 a dvě pomocná okna pro zobrazení vstupního snímku a výsledku detekce.

Po zadání adresy obrázku do textového pole a kliknutí na tlačítko "Načíst obrázek", se vstupní snímek otevře v novém okně 6.2. Následně je možnost si vybrat ze dvou tlačítek. První tlačítko „Lokalizovat šachovnici – PClines“ spustí modul detekce a lokalizace šachovnice založené na metodě paralelních souřadnic a Druhé tlačítko „Lokalizovat šachovnici – Houghlines“ spustí modul modul detekce a lokalizace šachovnice založené na Houghově transformaci. Dále proběhne detekce šachových figurek. Výsledkem je vizualizace predikovaného stavu šachovnice 6.3.



Obrázek 6.1: Hlavní okno demo aplikace.

Obrázek 6.2: Okno pro kontrolu načteného obrázku.



Obrázek 6.3: Výsledná predikce stavu šachovnice.

Kapitola 7

Experimenty a výsledky

Tato kapitola pojednává o experimentech, které vedly ke zvolení parametrů jednotlivých modulů, a provedení měření, která se zaměřila na přesnost lokalizace.

Pro zjištění ideálních parametrů, které vedou k nejlepším výsledkům modulů pro detekci a lokalizaci šachovnice, a měření výsledků je zapotřebí provést jeho validaci na testovací datové sadě. Testovací datová sada obsahuje zhruba dvanáct obrázků každého šachového setu. Obsahuje 1472 pozic figurek a 2830 mezer (celkový počet kontrolovaných pozic je 4302). Tyto obrázky jsou rozděleny na jednoduší a těžší. Snímek se považuje za těžší, pokud obsahuje více figurek, které zakrývají větší část šachovnice, nebo je horší kvalita snímku (například z důvodu šumu v pozadí).

Vzhledem k potřebě uživatelské interakce byl vytvořen pomocný skript v Pythonu *midPointAnnotation.py*, který slouží k anotování středů šachovnic. Na vstup dostává adresu složky se snímky a následně nimi prochází. Uživatel klikáním do středu šachovnice ukládá bod do souboru *název_obrázku_midpoint.txt*. Tento soubor se dále využívá v testovacích skriptech k automatizaci.

7.1 Experimenty s moduly detekce šachovnice

Za účely experimentování s parametry a měření výsledků byl vytvořen skript v Pythonu *evaluateChessboardDetection.py*, který implementuje veškerá měření na testovací datové sadě. Iterativně prochází snímky testovací sady, volá testovaný modul a zaznamenává výsledky.

PClines

Pro modul detekce a lokalizace šachovnice pomocí paralelních souřadnic je potřeba stanovit velikost akumulátoru d a prahu minimálních intenzit t. Experimenty byly zaměřené na získání kombinace parametrů s nejlepším možným výsledkem. Mezi měřené statistiky patří počet správných a chybných lokalizací, počet chyb detekce a průměrný čas běhu.

Menší a větší hodnoty prahu a velikosti akumulátoru vykazovaly ještě větší chybovost. V tabulce 7.1 jsou zmíněny jen relevantní hodnoty pro parametry. Nejlepších výsledků na testovací sadě dosahují parametry $t = 0,3$ a $d = 512$ s průměrnou přesností 97,1% a průměrnou dobou běhu 135 milisekund. Za zmínku také stojí kombinace parametrů $t = 0,35$ a $d = 256$, které dosahují přesnosti 96,1% s průměrnou dobou trvání 83 milisekund.

t	*d*	správná lokace	nesprávná lokace	chyby	prům. doba [s]
0,3	128	59	4	39	0,0775
	256	90	1	11	0,1136
	512	99	0	3	0,1353
0,35	128	53	3	46	0,0644
	256	98	0	4	0,0830
	512	89	0	13	0,1217
0,4	128	49	1	52	0,0475
	256	85	2	15	0,0678
	512	72	1	29	0,1186
0,45	128	39	1	62	0,0431
	256	71	3	28	0,0521
	512	49	3	50	0,1149

Tabulka 7.1: Výsledky experimentů modulu paralelních souřadnic.

Houghlines

Modul využívající Houghovu transformaci k detekci čar ovlivňují parametry minimální délky detekované čáry *min_length*, maximální vzdálenosti čar *max_gap* a prahu pro zvolení čar *thresh*.

Z výsledků vyplývá, že ideální hodnoty pro parametry jsou následující: $threshold = 60$ (tabulka 7.2), $minLineLength = 200$ (tabulka 7.3) a $maxLineGap = 100$ (tabulka 7.4). Časová náročnost se liší jen v rámci jednotek milisekund. Při volbě parametru minimální délky čáry byla zvolena hodnota podle minimálního počtu detekovaných čar, což má za následek snížení výpočetní náročnosti.

Porovnání výsledků detekce šachovnice

V tabulce 7.5 je přehled nejlepších výsledků modulů pro detekci šachovnice. Parametry jsou zapsány ve zkratkách, kde g je *maxLineGap*, l představuje *minLineLength* a t *threshold*. Můžeme si všimnout, že moduly dosahují stejné přesnosti, ale modul založený na Houghově transformaci je rychlejší o 90 milisekund. Tabulka také obsahuje příklad parametrů pro modul PClines, který má menší přesnost v rámci zrychlení výpočtu.

thresh	správná lokalizace	nesprávná lokalizace	chyby
40	68	4	31
50	85	1	16
60	99	0	3
70	97	0	5
80	96	0	6
90	92	0	10
100	81	1	20

Tabulka 7.2: Tabulka experimentů s prahem pro HoughLines.

min_length	správná lokalizace	nesprávná lokalizace	chyby
50	99	0	3
100	99	0	3
150	99	0	3
200	99	0	3
250	90	0	12

Tabulka 7.3: Tabulka experimentů s minimální délkou čar pro HoughLines.

max_gap	správná lokalizace	nesprávná lokalizace	chyby
80	92	0	10
90	94	0	8
100	99	0	3
110	98	0	4
120	97	0	5

Tabulka 7.4: Tabulka experimentů s maximální vzdáleností čar pro HoughLines.

	průměrná doba [s]	přesnost [%]	chybovost [%]
d = 256, t = 0,35	0,083	96,078	3,922
d = 512, t = 0,3	0,1353	97,058	2,941
g = 100, l = 200, t = 60	0,0451	97,058	2,941

Tabulka 7.5: Tabulka porovnání nejlepších výsledků modulů PClines a HoughLines.

PC	yolov4	yolov3	yolov4 tiny
správná lokalizace	74	60	59
nesprávná lokalizace	24	38	39
správná figurka	1360	1296	1294
nesprávná figurka	40	58	55
chyby lokalizace	32	95	89
průměrná doba [s]	0,37	0,347	0,192
celková přesnost [%]	72,5	58,8	57,8
přesnost lokalizace [%]	92,3	88	87,9

Tabulka 7.6: Porovnání modelů CNN s detekcí šachovnice paralelními souřadnicemi.

HT	yolov4	yolov3	yolov4 tiny
správná lokalizace	76	59	58
nesprávná lokalizace	23	40	41
správná figurka	1415	1343	1342
nesprávná figurka	39	59	59
chyby lokalizace	29	100	92
průměrná doba [s]	0,339	0,312	0,159
celková přesnost [%]	74,5	57,8	56,9
přesnost lokalizace [%]	96,1	91,2	91,2

Tabulka 7.7: Porovnání modelů CNN s detekcí šachovnice Houghovou transformací.

7.2 Výsledky predikce šachovnice

Za účelem celkového vyhodnocení úspěšnosti lokalizace šachové partie byl vytvořen skript v Pythonu *evaluateLocalization.py*. Tento skript iterativně prochází skrz testovací sadu snímků a vyhodnocuje výsledek modulů detekce a lokalizace. Zaměřuje se hlavně na přesnost a výpočetní náročnost. Na datové sadě byly testovány modely YOLOv4, YOLOv3 a jako experiment byl přidán model YOLOv4 tiny.

Než zhodnotíme výsledky, je potřeba objasnit, které parametry sledujeme a co znamenají. Správná lokalizace představuje úplnou shodu s testovacím obrázkem (generovaný FEN je shodný s testovacím FEN). Špatná lokalizace znamená, že došlo alespoň na jednom místě k chybě. Správná figurka představuje počet, kolikrát byla detekována a lokalizována správná figurka. Chybná lokalizace znamená, že figurka je v místě, kde nemá být, nebo figurka chybí. Průměrná doba představuje průměrný čas běhu programu na všech správně detekovaných šachovnicích. Skládá se z času potřebného pro detekci šachovnice a figurek.

Z tabulek 7.6 a 7.7 vyplývá, že nejlepších výsledků dosahuje kombinace YOLOv4 a detekce šachovnice založené na Houghově transformaci. Tato kombinace dosahuje přesnosti úplné lokalizace 74,5%, správnosti lokalizace figurek 96% a chybovosti 0,67%. Průměrná doba výpočtu je 339 milisekund, kdy většinu času využije CNN (většinou 290 až 320 milisekund). Z tohoto důvodu byl přidán experiment s YOLOv4 tiny, kde se průměrná doba výpočtu snížila o 200 milisekund, ale trojnásobně se zvětšila chybovost. Také se snížila přesnost úplné lokalizace o 17,6% a přesnost lokalizace figurek o 4,9%.

Kapitola 8

Závěr

Cílem práce bylo analyzova stav šachové partie a lokalizovat šachové figurky na šachovnici z fotografie. Tato úloha byla rozdělena na tři hlavní části - detekce šachovnice, detekce figurek a lokalizace.

K rozpoznání šachovnice jsem využil přístupu detekce čar. Tento přístup vychází z vědomosti, že šachovnice je přesně určená osmnácti čárami. Pro detekci šachovnice byly vytvořeny dva moduly. První modul je založen na detekci čar pomocí Houghovy transformace a druhý pomocí algoritmu PClines.

Detekce figurek byla realizována konvoluční neuronovou sítí. K trénování byla vytvořena datová sada, která obsahuje 1700 obrázků. Vzhledem k tomu, že neexistují skoro žádné datové sady této tématiky, většina fotek byla získána mnou a mými kamarády. Anotace datové sady byla uskutečněna díky vědomosti, že pozice figurek jsou jednoznačně určeny na šachovnici. Díky tomuto faktu stačilo zapsat pozice figurek a anotovat rohy šachovnice. Následně jsem vytvořil pomocí homografie a odhadu pozice kamery ohraničující kvádry políček a z nich ohraničující rámečky figurek. Celkem bylo vytvořeno téměř 20 tisíc anotací. Na této datové sadě jsem natrénoval modely YOLOv3, YOLOv4 a YOLOv4 tiny.

Lokalizace pracovala s výstupy předchozích částí. Výstup detekce šachovnice představuje mřížku středů políček a výstup detekce figurek jsou jejich ohraničující boxy. Spojením těchto výstupů vznikl řetězec FEN, který reprezentuje predikci aktuálního stavu šachovnice. Vzhledem k tématu práce byla vytvořena demo aplikace, která daný řetězec zobrazí v podobě SVG obrázku.

Moduly detekce šachovnice na testovací datové sadě dosahují přesnosti 97,1% viz. tabulka 7.5. Modely konvoluční neuronové sítě dosahují přesnosti detekce figurek až 96,1% viz. tabulka 7.7 řádek *přesnost lokalizace*. Celková přesnost lokalizace dosahuje hodnoty 74,5% řádek *celková přesnost*.

Prvotně měla být lokalizace šachovnice uskutečněna bez interakce s uživatelem, ale nedařilo se mi lokalizovat šachovnici. Problém jsem řešil předpokladem, že pokud uživatel bude chtít lokalizovat figurky na šachovnici, šachovnici bude zarovnávat na střed snímku. Toto řešení fungovalo dobře pro fotografie pod úhly blízké pohledu shora, ale výsledky nebyly podle mých představ. Z tohoto důvodu jsem se inspiroval předchozím řešením podobné tématiky a využil interakce s uživatelem. Dále mě napadlo využít detekce kontur k lokalizaci políček a vybrat políčko, které má ze všech čtyř stran přímku. Vzhledem k tomu, že šachovnice je mřížka, pomocí vyhledání dalších políček se dá, po určení dostatečného počtu, určit, kde se políčko nachází vůči šachovnici a následně pomocí homografie určit zbylé body. Bohužel jsem toto řešení nestihl implementovat.

Pokračování práce bych prvně směroval k úplné automatizaci. Chtěl bych vyzkoušet lokalizaci šachovnice pomocí detekce kontur. Zároveň vzhledem k výsledkům modelu YOLOv4 tiny, je dosti reálný nápad vytvořit mobilní aplikaci. Dalším možným pokračováním je integrace tzv. *chess engine* k získání nejlepšího tahu nebo ohodnocení tahu hráče.

Literatura

[1] Bapat, K. *Hough Transform with OpenCV (C++/Python)* [online]. Learn OpenCV, duben 2019 [cit. 2021-26-04]. Dostupné z: https://learnopencv.com/hough-transform-with-opencv-c-python/.

[2] Bochkovskiy, A., Wang, C.-Y. a Liao, H. YOLOv4: Optimal Speed and Accuracy of Object Detection. *ArXiv*. 2020, abs/2004.10934.

[3] Bradski, G. *OpenCV: Canny Edge Detector* [online]. Bradski, G., květen 2021 [cit. 2021-25-04]. Dostupné z: https://docs.opencv.org/3.4/da/d5c/tutorial_canny_detector.html.

[4] Bradski, G. *OpenCV: Hough Line Transform* [online]. Bradski, G., květen 2021 [cit. 2021-26-04]. Dostupné z: https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html.

[5] Canny, J. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1986, PAMI-8, č. 6, s. 679–698. DOI: 10.1109/TPAMI.1986.4767851.

[6] Chess.com. *Chess Notation & Algebraic Notation* [online]. 2020. 2021 [cit. 2021-20-04]. Dostupné z: https://www.chess.com/terms/chess-notation#whatiscn.

[7] Czyzewski, M., Laskowski, A. a Wasik, S. Chessboard and Chess Piece Recognition With the Support of Neural Networks. *Foundations of Computing and Decision Sciences*. Prosinec 2020, sv. 45, s. 257–280. DOI: 10.2478/fcds-2020-0014.

[8] Czyzewski, M., Laskowski, A. a Wasik, S. Chessboard and Chess Piece Recognition With the Support of Neural Networks. *Foundations of Computing and Decision Sciences*. Prosinec 2020, sv. 45, č. 4, s. 257–280. DOI: 10.2478/fcds-2020-0014.

[9] Ding, J. ChessVision : Chess Board and Piece Recognition. In: Ding, J., ed. *ChessVision: Chess Board and Piece Recognition*. 2016.

[10] Dubrofsky, E. a Woodham, R. J. Combining Line and Point Correspondences for Homography Estimation. In: Bebis, G., Boyle, R. D., Parvin, B., Koracin, D., Remagnino, P. et al., ed. *Advances in Visual Computing, 4th International Symposium, ISVC 2008, Las Vegas, NV, USA, December 1-3, 2008. Proceedings, Part II*. Springer, 2008, sv. 5359, s. 202–213. Lecture Notes in Computer Science. DOI: 10.1007/978-3-540-89646-3_20. Dostupné z: https://doi.org/10.1007/978-3-540-89646-3_20.

[11] Dubská, M., Herout, A. a Havel, J. PClines - Line Detection Using Parallel Coordinates. In: *Proceedings of CVPR 2011*. IEEE Computer Society, 2011, s. 1489–1494. ISBN 978-1-4577-0393-5.

[12] Edwards, S. J. *Standard: Portable Game Notation Specification and Implementation Guide* [online]. 1994 [cit. 2021-20-04]. Dostupné z: http://www.saremba.de/chessgml/standards/pgn/pgn-complete.htm.

[13] Escalera, A. De la a Armingol, J. M. Automatic Chessboard Detection for Intrinsic and Extrinsic Camera Parameter Calibration. *Sensors*. 2010, sv. 10, č. 3, s. 2027–2044. DOI: 10.3390/s100302027. ISSN 1424-8220. Dostupné z: https://www.mdpi.com/1424-8220/10/3/2027.

[14] Fisher R., W. A. a Wolfart, E. *Canny Edge Detector* [online]. 2003 [cit. 2021-25-04]. Dostupné z: https://homepages.inf.ed.ac.uk/rbf/HIPR2/canny.htm.

[15] Fisher R., W. A. a Wolfart, E. *Hough Transform* [online]. Fisher R., Perkins S., Walker A. and E. Wolfart, 2003 [cit. 2021-26-04]. Dostupné z: https://homepages.inf.ed.ac.uk/rbf/HIPR2/canny.htm.

[16] Hack, J. a Ramakrishnan, P. *CVChess: Computer Vision Chess Analytics* [online]. Stanford University, Stanford, 2014. Dostupné z: https://cvgl.stanford.edu/teaching/cs231a_winter1415/prev/projects/chess.pdf.

[17] Hartley, R. a Zisserman, A. *Multiple View Geometry in Computer Vision*. 2. vyd. Cambridge University Press, 2004. ISBN 9780511811685.

[18] Hollemans, M. *One-stage object detection* [online]. Machinethink, červen 2018 [cit. 2021-04-05]. Dostupné z: https://machinethink.net/blog/object-detection/.

[19] IBM Cloud Education. *What are Convolutional Neural Networks?* [online]. IBM, říjen 2020 [cit. 2021-04-05]. Dostupné z: https://www.ibm.com/cloud/learn/convolutional-neural-networks.

[20] jurgenwesterhof. *Chess Pieces Sprite* [online]. Září 2014 [cit. 2021-03-05]. Dostupné z: http://commons.wikimedia.org/wiki/Template:SVG_chess_pieces.

[21] Juránek, R. *Pclines-python* [online]. GitHub, 2020. Dostupné z: https://github.com/RomanJuranek/pclines-python.

[22] Kacmajor, T. *Hough Lines Transform Explained* [online]. Medium, červen 2017 [cit. 2021-28-04]. Dostupné z: https://medium.com/@tomasz.kacmajor/hough-lines-transform-explained-645feda072ab.

[23] Kapoor, S. *An introduction to Epipolar Geometry* [online]. Sanyam Kapoor, srpen 2017 [cit. 2021-20-04]. Dostupné z: https://im.perhapsbay.es/kb/an-introduction-to-epipolar-geometry#fnref-@hartley2003multiple.

[24] Karri, S. T. *Hough Transform* [online]. Medium, září 2019 [cit. 2021-26-4]. Dostupné z: https://medium.com/@st1739/hough-transform-287b2dac0c70.

[25] Kathuria, A. *What's new in YOLOv3?* [online]. towards data science, duben 2018 [cit. 2021-07-05]. Dostupné z: https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b.

[26] KLUBOVNA Česká. *ČESKÁ KLUBOVKA* [online]. ČESKÁ KLUBOVKA, 2017 [cit. 2021-20-04]. Dostupné z: http://sachy.csla.cz/.

[27] KORAY, C. a SÜMER, E. A Computer Vision System for Chess Game Tracking. In: *21st Computer Vision Winter Workshop, Rimske Toplice, Slovenia*. 2016. ISBN 978-961-90901-7-6.

[28] LEE, T.-H. H. a TAIPEI, T. R. Edge detection analysis. *IJCSI International Journal of Computer Science Issues*. 1. vyd. 2007, sv. 5, č. 6.

[29] LEPETIT, V., MORENO NOGUER, F. a FUA, P. EPnP: An accurate O(n) solution to the PnP problem. *International Journal of Computer Vision*. 1. vyd. Únor 2009, sv. 81, č. 155. DOI: 10.1007/s11263-008-0152-6. ISSN 0920-5691.

[30] LU, X. X. A Review of Solutions for Perspective-n-Point Problem in Camera Pose Estimation. *Journal of Physics: Conference Series*. 1. vyd. IOP Publishing. sep 2018, sv. 1087, č. 5, s. 052009. DOI: 10.1088/1742-6596/1087/5/052009. Dostupné z: https://doi.org/10.1088/1742-6596/1087/5/052009.

[31] MISCELLANEOUS, E. *FIDE Handbook E. Miscellaneous* [online]. 2018 [cit. 2021-20-04]. Dostupné z: https://handbook.fide.com/chapter/E012018.

[32] MORDVINTSEV, A. a K., A. *Canny Edge Detection* [online]. Alexander Mordvintsev and Abid K., 2013 [cit. 2021-25-4]. Dostupné z: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html.

[33] MÜLLER, M. *The Perspective-n-Point problem* [online]. Duben 2020 [cit. 2021-11-05]. Dostupné z: https://www.researchgate.net/figure/The-Perspective-n-Point-problem-is-stated-as-the-issue-of-estimating-an-image-pose_fig4_342379262.

[34] ORAC, R. *What's new in YOLOv4?* [online]. towards data science, květen 2020 [cit. 2021-07-05]. Dostupné z: https://towardsdatascience.com/whats-new-in-yolov4-323364bb3ad3.

[35] REDMON, J. *Darknet: Open Source Neural Networks in C* [online]. 2013–2016. Dostupné z: http://pjreddie.com/darknet/.

[36] REDMON, J., DIVVALA, S., GIRSHICK, R. a FARHADI, A. You Only Look Once: Unified, Real-Time Object Detection. In: REDMON, J., ed. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, s. 779–788. DOI: 10.1109/CVPR.2016.91. ISBN 978-1-4673-8852-8.

[37] REDMON, J. a FARHADI, A. YOLOv3: An Incremental Improvement. *ArXiv*. 1. vyd. 2018, abs/1804.02767, č. 1.

[38] REPUBLIKY Šachový svaz České. *1. lekce: Úvod. Šachovnice a označení polí* [online]. Šachový svaz České Republiky, říjen 2017 [cit. 2021-20-04]. Dostupné z: https://www.chess.cz/1-lekce-uvod-sachovnice-a-oznaceni-poli/.

[39] REPUBLIKY Šachový svaz České. *2. lekce: Šachové figury a jejich pohyb* [online]. Šachový svaz České Republiky, 2017 [cit. 2021-20-04]. Dostupné z: https://old.chess.cz/www/mladez/metodika/zakladni-sachovy-vycvik/2_lekce.html.

[40] Roboflow. *Chess Pieces Dataset* [online]. Roboflow, únor 2021 [cit. 2021-08-05]. Dostupné z: https://public.roboflow.com/object-detection/chess-full.

[41] Szeliski, R. *Computer Vision: Algorithms and Applications*. 2. vyd. Springer London, 2010. Texts in Computer Science. ISBN 9781848829350. Dostupné z: https://www.dropbox.com/sh/88qvr1z7fpfx1tv/AAB4Ia3yEMuZ4WSzNWB5acTta?dl=0&preview=SzeliskiBookDraft_20210327.pdf.

[42] team, S. image. *Hough Transform* [online]. Scikits-image team, 2011 [cit. 2021-26-4]. Dostupné z: https://scikit-image.org/docs/0.6/auto_examples/plot_hough_transform.html.

[43] Varun, R. a Gupta, S. Chess recognition using computer vision. *The Australian National University, Canberra, Australia*. 1. vyd. 2017, č. 1.

[44] Weitz, D. *Parallel Coordinates Plots* [online]. towards data science, leden 2020 [cit. 2021-28-04]. Dostupné z: https://towardsdatascience.com/parallel-coordinates-plots-6fcfa066dcb3.

[45] Zhang, Z. A Flexible New Technique for Camera Calibration. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*. 1. vyd. Prosinec 2000, sv. 22, č. 5, s. 1330 – 1334. DOI: 10.1109/34.888718.

[46] Zucker, M. *Unprojecting text with ellipses* [online]. Říjen 2016 [cit. 2021-11-05]. Dostupné z: https://mzucker.github.io/2016/10/11/unprojecting-text-with-ellipses.html.

Příloha A

Obsah přiloženého paměťového média

```
/
├── README.txt
├── images.zip
├── video.mp4
├── /src
│   ├── boardAnotation.py
│   ├── boundingBoxes.py
│   ├── chessboardDetectionHoughlines.py
│   ├── chessboardDetectionPClines.py
│   ├── demo.py
│   ├── figureAnotation.py
│   ├── figureDetection.py
│   ├── midPointAnotation.py
│   ├── plotAvgLoss.py
│   ├── yolov3.cfg
│   ├── yolov3.weights
│   ├── yolov4.cfg
│   ├── yolov4.weights
│   ├── yolov4-tiny.cfg
│   └── yolov4-tiny.weights
└── /latex
    ├── /src
    └── /pdf
```