Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Rozpoznání Rubikovy kostky v obraze kamery pomocí knihovny OpenCV

Bc. Libor Machovec

Diplomová práce

2019

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2018/2019**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Libor Machovec**
Osobní číslo:	**I17212**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Rozpoznání Rubikovy kostky v obraze kamery pomocí knihovny OpenCV**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

Cílem diplomové práce je inovovat mobilní aplikaci pro systém Android, která automaticky rozpozná a zaznamená Rubikovu kostku z obrazu kamery. Součástí bude navržení vlastního algoritmus pro zpracování jednotlivých snímků kamery, který bude detekovat jednotlivá políčka Rubikovy kostky a jejich pozice. V teoretické části budou analyzovány jednotlivé postupy pro detekci objektů na snímku. Navržené postupy budou vyhodnoceny z hlediska přesnosti a rychlosti. V praktické části bude algoritmus implementován do aplikace pro operační systém Android. Závěrem budou navržena a zdokumentována zlepšení představené aplikace a postup vývoje.

Rozsah grafických prací: -

Rozsah pracovní zprávy: **min. 30 stran**

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

BRADSKI, G., KAEHLER, A. Learning OpenCV: Computer Vision with the OpenCV Library. 1. vyd. O'Reilly Media, 2008, s. 14. ISBN 978-05-9651-613-0 CAVALLARO, A. Image analysis and computer vision for undergraduates. In: 2005 IEEE International Conference on Acoustics, Speech, and Signal Processing. Philadelphia, Pennsylvania, USA: The Institute of Electrical and Electronics Engineers Signal Processing Society, 2005, s. 577 HORÁK, K., KALOVÁ, I. Počítačové vidění - Počítačová cvičení (64087) - VUT v Brně (online). [cit. 2012-10-13]. URL JETENSKÝ, P. Zdrojové kódy - Computer vision course with OpenCV (online). [cit. 2012-10-13]. URL

Vedoucí diplomové práce: **Ing. Pavel Jetenský, Ph.D.**
Katedra informačních technologií

Datum zadání diplomové práce: **22. října 2018**

Termín odevzdání diplomové práce: **18. května 2019**

Ing. Zdeněk Němec, Ph.D.
děkan

L.S.

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 17. listopadu 2018

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 9. 5. 2019

Bc. Libor Machovec

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu práce Ing. Pavlu Jetenskému, Ph.D. za odborné rady a pomoc při práci na této Diplomové práci. Dále bych rád poděkoval kolegům studentům, na které jsem se mohl obrátit s veškerými dotazy.

ANOTACE

Tato Diplomová práce popisuje a dokumentuje vytvoření aplikace pro mobilní zařízení, která uživateli bude pomáhat při skládání Rubikovi kostky. Důraz byl kladen na problematiku digitálního zpracování obrazu pomocí kamery mobilního zařízení. Díky tomu aplikace automaticky zaznamená Rubikovu kostku a na základě digitálního zpracování obrazu pomocí navrženého algoritmu nalezne její řešení.

Tento dokument nejdříve popisuje zpracování obrazu a využité algoritmy pro snímání Rubikovy kostky. V další části je popsána výsledná aplikace z pohledu funkčnosti a uživatelského ovládání. V poslední části je popsána implementace a datové struktury využity při vývoji.

KLÍČOVÁ SLOVA

kamera, android, hlavolam, rozpoznání, OpenCV, aplikace, Java

TITLE

Recognizing the Rubik's Cube in-picture camera using OpenCV library

ANNOTATION

This Diploma Thesis describes and documents the creation of a mobile device application that will assist the user in solving the Rubik's cube. Emphasis was placed on digital image processing using a camera of a mobile device. As a result, the application automatically records the Rubik's cube and finds a solution based on the digital image processing using the proposed algorithm.

This document first describes image processing and algorithms used to capture Rubik's cube. The next part describes the resulting application in terms of functionality and user control. The last part describes implementation and data structures used in development.

KEYWORDS

camera, android, puzzle, recognition, OpenCV, app, Java

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

Seznam zkratek

3D	trojrozměrný, trojdimenzionální
HSV	Hue Saturation Value
CFOP	Cross, F2L, OLL a PLL
F2L	First two layers
OLL	Orient last layer
PLL	Permutate the last layer
JSON	JavaScript Object Notation
RGBA	Red Green Blue Alfa
LAB	Lightness A part B part
HW	hardware

ÚVOD

Rozpoznání objektů v obraze kamery je velice moderní způsob zaznamenávání informací o pozorovaném objektu. Tato práce popisuje jednu z možných implementací zpracování získaných informací tímto způsobem. Hlavním cílem práce je navrhnout takový algoritmus, který bez akce uživatele rozpozná objekt na snímku a zaznamená ho do definovaných struktur a objektů k dalšímu využití. Algoritmus bude vyvíjen a testován pro platformu Android. Tedy určení algoritmu bude primárně pro mobilní zařízení, jako je chytrý telefon nebo tablet. Objektem detekce bude Rubikova kostka, přesněji je to jedna stěna tohoto hlavolamu. Získané informace po načtení stěny Rubikovi kostky budou dále využity pro nový algoritmus, který bude hledat řešení pro složení Rubikovy kostky.

Tato práce tématem navazuje na Bakalářskou práci autora **Rozpoznání a složení Rubikovy kostky**. [1] V průběhu této práce budou komentovány řešení z Bakalářské práce. První kapitola popisuje a komentuje hotové řešení Bakalářské práce. Jsou zde popsány nedostatky a návrhy na zlepšení použitých algoritmů a metod.

Další část práce se věnuje problematice zpracování obrazu. Obraz bude zpracováván za pomoci knihovny **OpenCV**[1]. Bude zde rozveden postup budování algoritmů a způsob testování těchto algoritmů.

Práce bude také popisovat výslednou aplikaci. Zejména z pohledu uživatelského rozhraní a funkčnosti. Zde budou popsány navrhnuté datové struktury a jejich význam.

Závěrem bude popsáno porovnání výsledků nově navržených algoritmů s výsledky algoritmů Bakalářské práce.

[1] Open Source Computer Vision Library

1 CÍL PRÁCE

Cílem této Diplomové práce je navrhnout a implementovat vylepšení a potlačit nedostatky stávající aplikace Bakalářské práce. Výstupem Bakalářské práce byla aplikace pro platformu Android. Koncept aplikace je převeden do této práce. Zejména je dodrženo použití dvou uživatelských obrazovek. Tady ale veškerá podobnost končí. [1]

Hlavní cíl je automatické načtení Rubikovy kostky. Bude navržen takový algoritmus, který bude zpracovávat jednotlivé snímky za účelem rozpoznání jednotlivých políček Rubikovy kostky, a to vše v reálném čase. Druhým cílem je implementace lepšího algoritmu hledání řešení s důrazem na menší počet kroků.

1.1 Rozpoznání stěny kostky

První obrazovka aplikace Bakalářské práce sloužila pro načítání Rubikovy kostky. Načítání kostky bylo pro uživatele celkem náročné. Proces načtení vyžadoval pozornost a pečlivost uživatel. Chyběla animace nebo jiný intuitivní prvek, který by prezentoval potřebnou změnu orientace kostky pro další postup při načítání. V této fázi vznikalo poměrně mnoho chyb, které vedly k potřebné další korekci.

Kroky korekce byly velice časově náročné. Aplikace pouze textovou podobou informovala o potřebném natočení kostky. Samotné načítání probíhalo po jednotlivých stěnách. Stěnu kostky při snímání bylo nutné přesně umístit v požadované vzdálenosti na požadované místo na snímku. Následoval odhad barvy políčka kostky. Tento odhad spočíval v přiřazení barvy políčka do předem definovaného rozmezí pro každou barvu na kostce. Pokud byl uživatel s návrhem odhadnutých barev spokojen, potvrdil ukončení kroku načtení stěny tlačítkem.

Tato práce si klade za cíl navrhnout takový algoritmus, který rozpozná stěnu kostky v obraze a vhodným způsobem ji zaznamená k dalšímu užití.

1.2 Možná nestandartní rozložení kostky

Původní řešení snímání stěny kostky v Bakalářské práci špatně reagovalo na měnící se okolní světelné podmínky a také nebylo možné načítat kostky s jiným barevným rozložením nebo dokonce s úplně odlišnými barvami od standardních barev stěn Rubikovi kostky (bílá, žlutá, modrá, zelená, červená, oranžová). [1]

Algoritmus by měl být schopen ze zaznamenaných stěn sestrojit model kostky s aktuálním rozložením jednotlivých políček na konkrétních stěnách. Toto řešení by nemělo být závislé na okolních světelných podmínkách ani na barevné paletě celé kostky. Dále bude doplněna komponenta např. model kostky, která bude nejen ukazovat již zaznamenaný postup, ale také bude intuitivně zobrazovat potřebné kroky jako je otočení celé kostky.

1.3 Optimalizace funkce editace modelu

Druhá část Bakalářské práce řešila validaci načteného modelu, hledání řešení dalších kroků, prezentaci řešení a prezentaci celkového modelu. Výsledek validace byl předán uživateli pouze v textové podobě, kde byly shrnuty nedostatky načteného modelu (chyby v počtu barev, neexistující kombinace na jednotlivém dílku, apod.). Uživatel pro další krok byl nucen veškeré nedostatky odstranit.

Odstranění nedostatků bylo prováděno manuálním nastavením barev na jednotlivých políčkách. Vzhledem k celkem nepřesnému způsobu načítání tento krok byl vynucen téměř vždy. Navíc se zpravidla jednalo o chyby na více než jednom políčku. V této práci bude odstraněn prvek editace modelu kostky. To bude umožněno díky použití skoro dokonalého načítání, při kterém není nutné model nijak opravovat.

1.4 Menší počet kroků řešení

Následné nalezení řešení bylo prezentováno pomocí dialogu. Uživatel viděl pouze název potřebného kroku a obrázek složené kostky se znázorněním kroku. To vedlo k mnoha nepřesnostem a chybám ze strany uživatele. A následně tyto chyby vedly k neúspěšnému dokončení složení kostky. Celý proces bylo nutné opakovat. Výsledný počet kroků pak byl zpravidla okolo 170 i více kroků. [1]

Pro prezentaci řešení a modelu bude zahrnut 3D model kostky, na kterém bude jasně vidět aktuální stav skládání a budou zde potřebné akce přehledně prezentovány pomocí animace. Hledání řešení bude optimalizováno za účelem snížení počtu potřebných kroků.

2 ROZPOZNÁNÍ OBRAZU

Rozpoznání obrazu je jedním z mnoha způsobů sběru a digitalizace informací. Pro načtení objektu, jako je Rubikova kostka, je kamera mobilního telefonu (tabletu) více než vhodná. Z načtených dat lze zjistit informace o barvě **jednotlivých políček**[2]. Také lze určit vzájemnou polohu jednotlivých políček. Postupným načtením všech 6 stěn lze zpětně sestrojit model kostky. Pro usnadnění zpracování obrazu je výhodné použít knihovnu nebo knihovny, které podporují tuto disciplínu. Velice oblíbenou knihovnou je knihovna **OpenCV**.

Knihovna OpenCV je vyvíjena za účelem zpracování obrazu a strojového učení. Knihova obsahuje více než 2500 optimalizovaných algoritmů sestávajících z široké škály matematických metod nebo i jednoduchých algoritmů určených například k vykreslování, či modifikaci obrazu, načítání a ukládání souborů. Knihovna je často využívána k rozpoznávání tváří, identifikaci objektů, sledování pohybu objektů ale také třeba snímající kamery. Další užití knihovny je např. při práci s 3D prostředím a modely obecně. Dále nabízí algoritmy pro korekci a úpravy fotek. V neposlední řadě podporuje vykreslování rozšířené reality. [2]

Dalším argumentem vhodnosti užití knihovny OpenCV je i to, že knihovnu využívají světově významné společnosti, jako je například: Google, Microsoft, Intel, IBM, Sony a Toyota. [2]

Rozhraní zvolené knihovny je pro tyto účely také plně vyhovující. Knihovna podporuje rozhraní pro technologie C++, Python, Java a MATLAB na široké základně platforem jako Linux, Android, Mac OS a Windows. Celá knihovna je psána nativně v jazyce C++. [2]

2.1 Teorie zpracování obrazu

Kamera snímá okolní obraz a následně ho vyjadřuje pomocí digitálních informací uložených v matici bodů. Tyto matice nazýváme snímky. Většina snímků je tvořena množinou pixelů. Pixel je elementární hodnota obrazu popisující jeden bod. Každý pixel tedy nese informaci sám za sebe. Pro získání více informací je vhodné zkoumání a testování pixelů vzájemně. Vzájemně lze zkoumat pixely ležící v jednom snímku, nebo napříč několika snímky. Výsledkem takto zpracovaného obrazu je v tomto případě popis jedné stěny Rubikovy kostky. A to zejména poloha jednotlivých políček na stěně.

[2] Rozumí se elementární část kostky jedné barvy

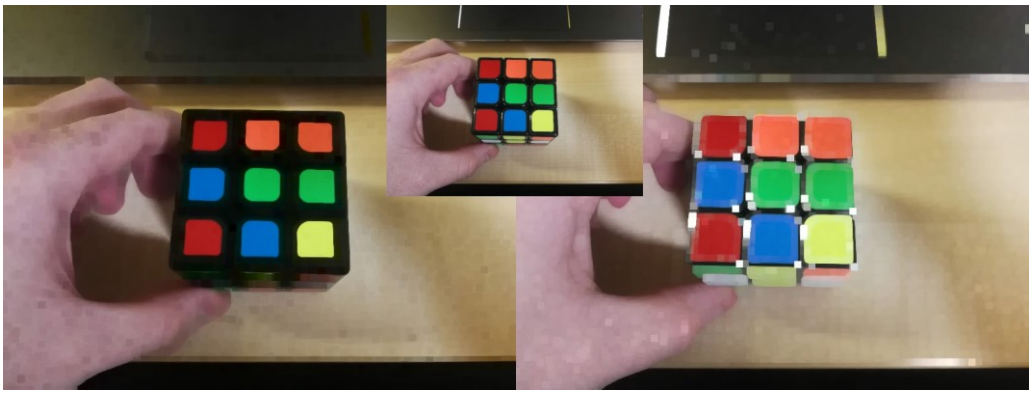
Toto zpracování je tvořeno zpravidla několika po sobě jdoucími kroky popsanými dále. V těchto kapitolách je obecně popsán problém, dostupné možnosti řešení problému a jak je výsledné řešení použito v aplikaci. K výslednému řešení bylo dospěno na základě testování algoritmů na sadě necelých 100 testovacích snímků. Mezi různými algoritmy poté byly vybrány ty s největší přesností (úspěšností). Vybrané algoritmy následně byly použity pro funkčnost tvořené aplikace.

2.1.1 Předzpracování obrazu

V této fázi je snaha ze snímku odebrat nebo upravit informace, které by mohly snižovat účinky nebo úplně znemožňovat pokračování v následném zpracování obrazu. V tomto smyslu se nejčastěji jedná o metody odstraňování šumu spojené s metodami konverze snímků z barevného spektra na černobílé spektrum. Odstraněním šumu lze dosáhnout spojitějších informací, což se projevuje například u linie obrysu obrazu. Při odstranění barev z obrazu lze také ušetřit potřebný výkon pro zpracování. Zároveň je zachována dostatečná informace pro nalezení například obrysů. Potřebný výkon je šetřen snížením hodnot na 1/3 původního množství (3 složky barevného snímku nahradí 1 složka intenzity černobílého snímku).

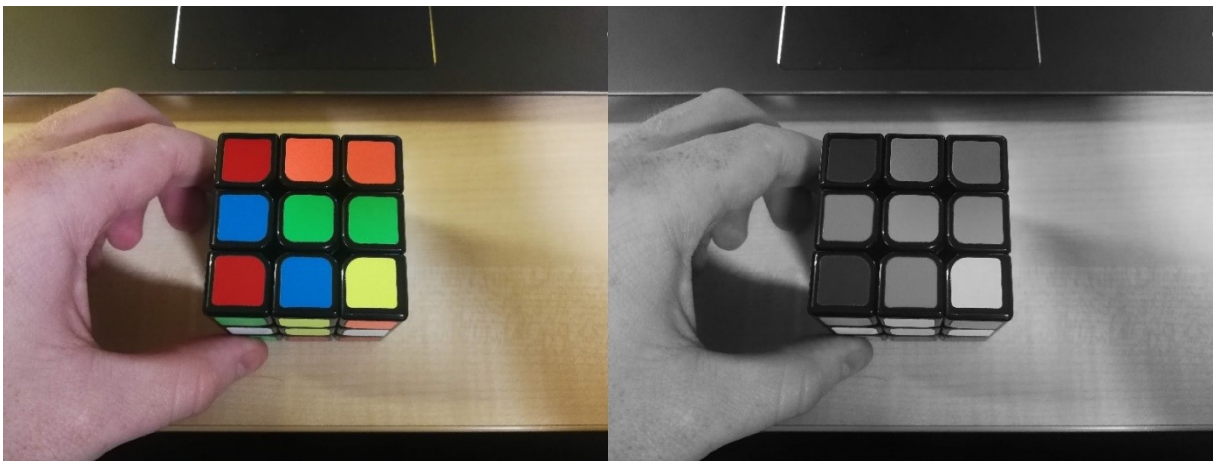
Knihovna OpenCV nabízí celou sadu funkcí pod souhrnným názvem `Image Filtering`. Nejpoužívanější metody odstranění šumu vytvářejí rozmazaný efekt. To je dáno způsobem přepočtu zdrojových dat na výsledná data, kdy dochází podle zadané funkce k výpočtu hodnoty na základě určené oblasti okolních hodnot. Využívají se zde funkce průměru, váženého průměru a doplnění dalších podmínek např. maximální hodnota. [3]

Dále pro předzpracování obrazu lze využít funkce geometrické transformace jako `Erosion` a `Dilation`. Funkce `Erode` vypočítává efekt smrštění a inverzní funkce `Dilate` naopak roztažení. Jak lze pozorovat na ukázce níže. Vlevo výstup funkce `Erode`, vpravo výstup funkce `Dilate` a ve středu originál. [3]



Obrázek 1: Ukázka výstupů funkcí `Erode` a `Dilate`

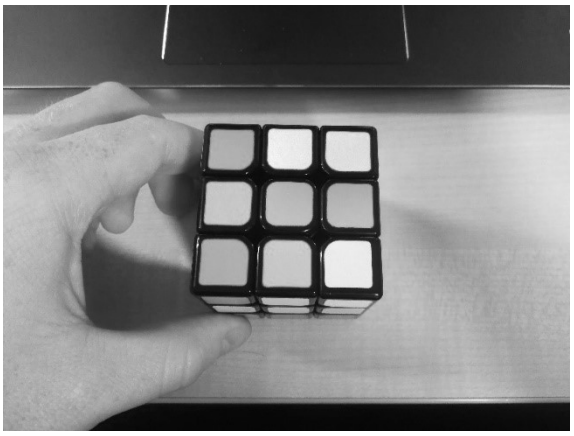
Pro cílový algoritmus je výsledek předzpracování snímku snad nejzásadnější. Cílem je získat binární snímek[3] ze zdroje. Zásadní je, aby na snímku byly co nejpřesněji stanoveny oblasti, kde se nachází jednotlivá políčka kostky. Z konstrukce kostky lze odvodit, že je zapotřebí rozdělit pixely snímku na ty které náleží černým (tmavým) okrajům a na ostatní nečerné pixely. Proto se nabízí zanedbat barevnou informaci a pracovat s černobílým snímkem ukázka níže.



Obrázek 2: Výsledek převedení snímku na odstíny šedi

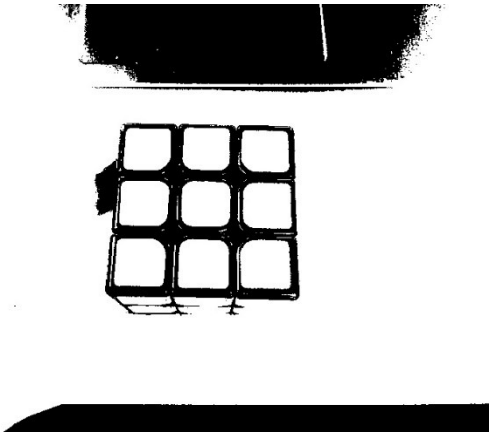
Z výsledku lze pozorovat nedostatečné oddělení tmavých barev jako je červená od okrajů kostky. Lepších výsledků lze dosáhnout využitím barevného modelu HSV[4]. U tohoto modelu je každá černá (tmavá) barva jednoznačně rozlišena ve složce V neboli hodnota. Extrahováním této složky lze dosáhnout výsledku níže.

[3] Binární snímek pro každý bod nese pouze informaci 1 nebo 0 (černá nebo bílá, pravda nebo nepravda)
[4] HSV – barevný model založen na složkách odstínu sytosti a hodnoty.



Obrázek 4: Extrahovaná hodnota V

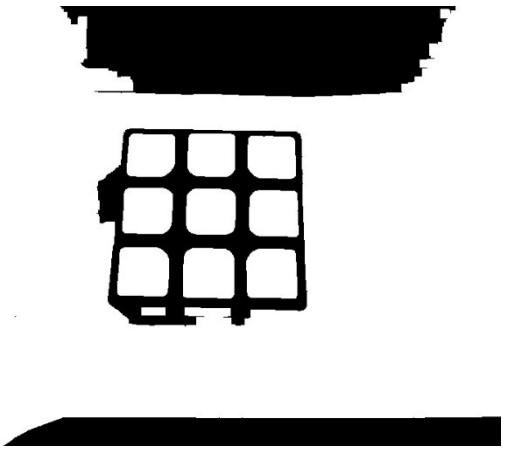
Takto získané hodnoty zbývá převést na zmiňovaný binární snímek. Tohoto lze dosáhnou jednoduchou funkcí prahové hodnoty `Threshold`. Funkce přiřazuje hodnotám menším jak práh 0 jinak maximální hodnotu. Výsledek aplikace na následujícím obrázku.



Obrázek 3: Výsledek funkce `Threshold`

Výsledek funkce `Threshold` stále není dokonalý. Obrysy jednotlivých políček obsahují „trhliny“, které při dalším zpracování generují zbytečné množství obrysů a tím zvyšují nároky na další zpracování a možnost chyby. Řešením jsou funkce `Erode` a `Dilate`, kterými lze efektivně omezit tyto nedokonalosti při zachování potřebné informace ve snímku. Funkce `Erode` „zaplní“ nedokonalosti a následná funkce `Dilate` vrátí tvary do původních rozměrů.

OpenCV nabízí funkci `MorphologyEx`. Tato funkce zabezpečuje popsaný postup. Následující výsledek po použití předchozích funkcí je již dostatečně přesný pro další zpracování.



Obrázek 5: Výsledek předzpracování snímku

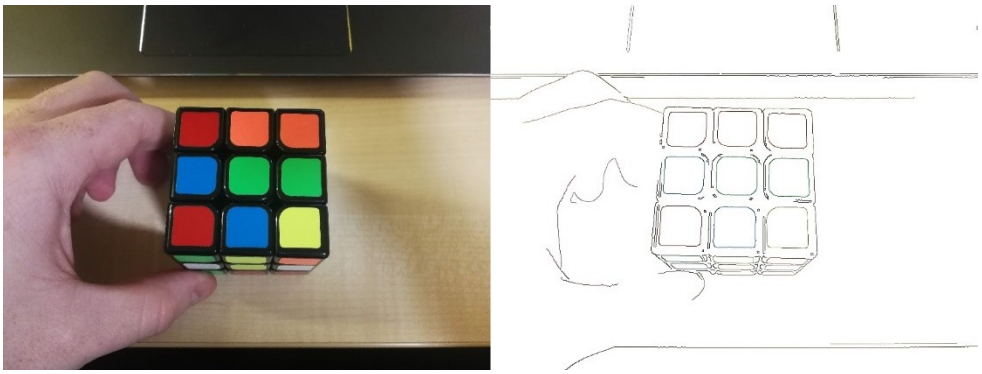
2.1.2 Metody nalezení obrysů

Metodou nalezení obrysů se označuje postup hledání skokových změn ve snímku. Skokovými změnami se nejčastěji projevují obrysy objektů. Nejznámější metodu hledání obrysů vynalezl **John Canny** v roce 1986 pod názvem **Canny Edge detector**. Implementace této funkčnosti je v metodě knihovny `Canny`. Jedná se o více fázový postup hledání hran.

Vstupem je obyčejný (základní) snímek.

- Následně je ze snímku odstraněn šum pomocí Gaussian filtru. Následně jsou hledány hrany neboli změny v obraze. Tento krok je proveden celkem 4x. Jsou hledány hrany vedoucí horizontálně, vertikálně a v diagonálních směrech. Zde se využívá derivací popsaných dále.
- Dalším krokem je potlačení nebo zesílení hran. Zde je využito sledování směru nalezení hrany. Posilují se hrany, které pokračují naopak neočekávané vychýlení je potlačeno.
- Dalším krokem je prahový filtr. Konkrétně dvojitý prahový filtr. Je zde zanedbána taková hrana, která svojí intenzitou nedosahuje požadované hranice. Může se jednat o barevné přechody jednotlivých barev nebo pozůstatek rušení.

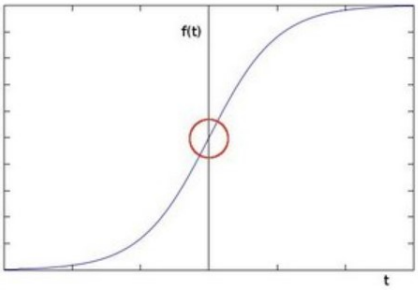
- Posledním krokem je spojení nalezených hran. Jedná se zejména o sledování hran. Pokud je hrana přerušena například pouze v jednom bodě a existuje hrana která pokračuje stejným směrem jsou tyto hrany spojeny a tedy posíleny. [4]



Obrázek 6: Výsledek funkce `Canny`

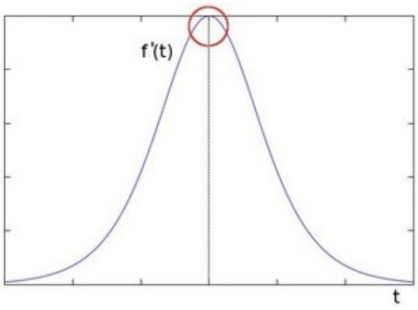
V předzpracovaném snímku jsou veškeré hrany značně posíleny, odstraněny nežádoucí rušení a nepřesnosti. Knihovna OpenCV nabízí celou řadu funkcí pro zpracování a určení struktur a tvarů na snímku. Dokumentace tyto funkce řadí do sekce `Structural Analysis and Shape Descriptors`.

Pro hledání obrysů je využito matematických operací například derivace, funkce `Sobel` vypočítá derivaci snímku. Pokud uvažujeme snímek jako jednorozměrné pole mohou data tvořit následující křivku.



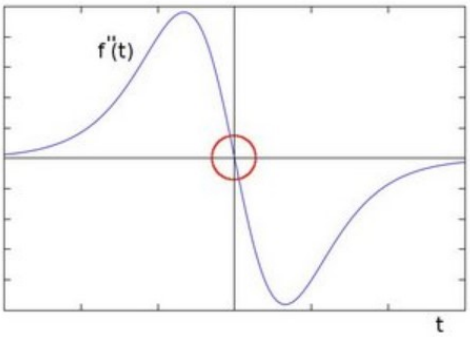
Obrázek 7: Možná reprezentace hrany ve snímku převzato z docs.opencv.org

Červený kruh označuje místo výskytu hrany neboli skokové změny informace. Derivací dat tohoto průběhu lze dosáhnou posílení neboli vyznačení takových přechodů.



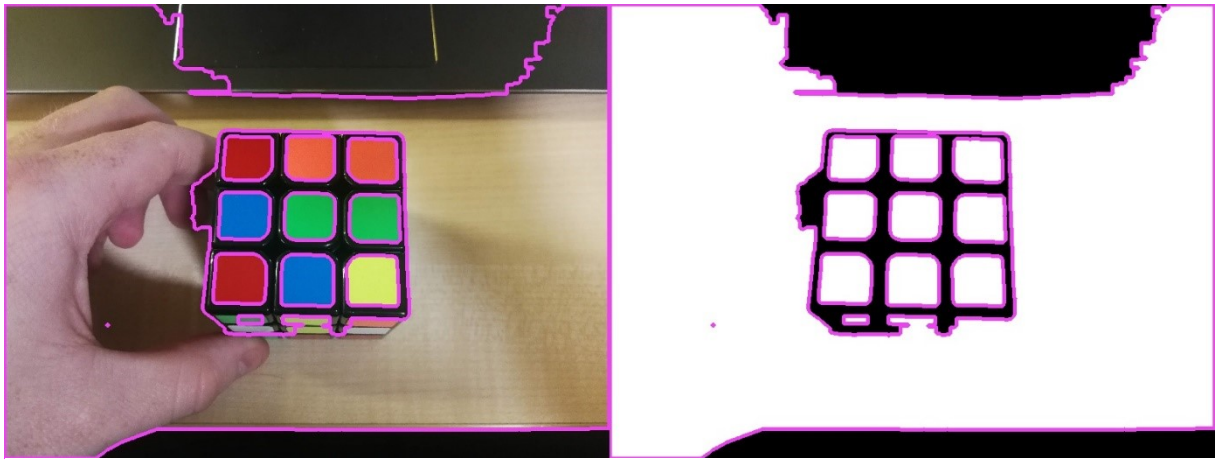
Obrázek 8: Derivace hrany převzato z docs.opencv.org

Výsledkem je tedy posílení všech takových to změn ve snímku, a tedy zvýraznění potencionálních obrysů. Pokud by takto připravený výsledek nebyl vhodný lze využít druhé derivace. V knihovně je tato funkce označována jako `Laplacian`.



Obrázek 9: Druhá derivace hrany převzato z docs.opencv.org

Funkce `FindContours` analyzuje snímek postupným následováním skokových změn, které znázorňují obrysy. Po vykreslení jednak do zdrojového snímku tak do předzpracovaného snímku sledujeme úspěšné nalezení potřebných obrysů.



Obrázek 10: Výsledek hledání obrysů

2.1.3 Roztřídění obrysů

Po úspěšném nalezení obrysů je nutné obrysy roztřídit na potencionální obrysy jednotlivých políček. Z geometrie jedné stěny Rubikovi kostky vyplývají následující fakta, ze kterých lze vycházet v dalších krocích. Jsou to fakta:

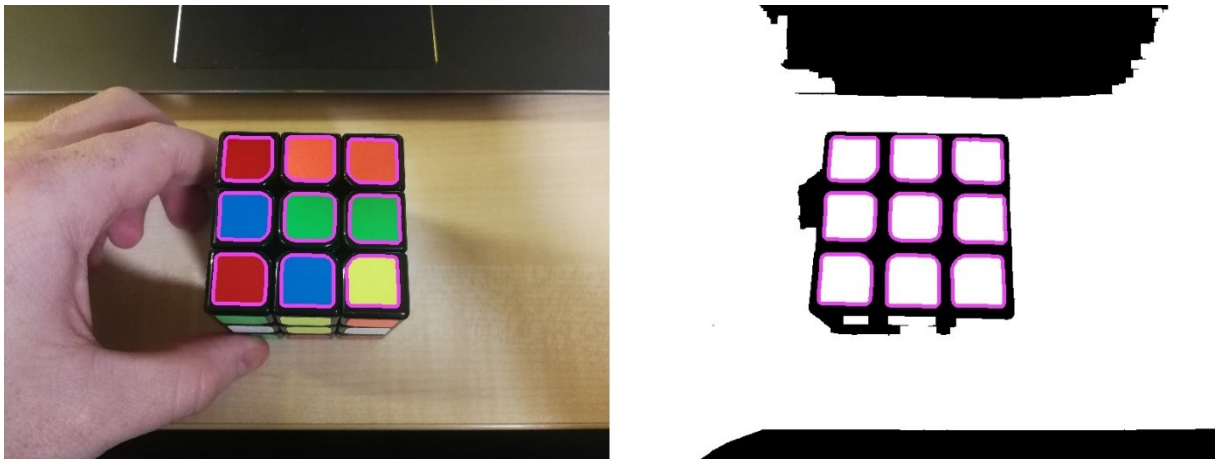
- Velikost (rozměr i obsah) políček je přibližně stejná;
- Jedná se o stejně orientované čtverce;
- Jsou umístěny v maticovém uspořádání 3x3;
- Nemají vnořené obrysy.

Využití těchto předpokladů slouží k sestavení třídícího algoritmu. Z velikostního předpokladu lze bezpečně určit maximální plocha jednoho políčka jako druhá mocnina jedné třetiny nejmenšího rozměru snímku. Minimální plochu lze stanovit stejným výpočtem uvažujícím minimální rozměr snímku jako 15 % tohoto rozměru.

Jestli se jedná o tvar připomínající čtverec lze určit faktorem kruhovitosti, který je dán vztahem:

$$\frac{4*\pi*obsah}{obvod^2}.$$

Výsledkem je koeficient rovný 1 jedná-li se o perfektní kruh pro čtverec je koeficient přibližně 0,78. Aplikováním následující pravidel dojde k odfiltrování obrysů, které jednoznačně nereprezentují jednotlivá políčka.



Obrázek 11: Filtrované obrysy

2.1.4 Vzájemná poloha obrysů

K odstranění vnořených tvarů do detekovaných políček (může se jednat o odlesky) je využito dalšího výsledku použité funkce `FindContours`, která vrací hierarchii zanoření jednotlivých obrysů. U obvodových obrysů je nadřazený obrys označen -1. Pro určení vzájemných poloh je využito vlastnosti maticového uspořádání. V tomto uspořádání natočení jednoho políčka určuje pozici ostatních, stejně jako velikost takového políčka. K určení natočení byla použita funkce `MinAreaRect`, která určuje 4 body nejmenšího opsaného obdélníku. Ze obdélníku lze určit natočení vůči ose x a délka hran obdélníku.

Určení sousedních políček je závislé na středu středového políčka, předpokládané vzdálenosti středu sousedního políčka a úhlu natočení oproti rovině umístěné matici. Pokud předpokládaný střed sousedního políčka leží v jednom z detekovaných obrysů.

Tento obrys je nastaven jako sousední na dané pozici. Tento postup je opakován pro všechny nalezené obrysy. Obrys, který nalezne všech 8 sousedních je nastaven na středový a zároveň je znám vztah k sousedním obrysům.

Obrázek 12: Výstup nalezených sousedních políček

2.1.5 Celkový popis obrysů

Tato fáze začíná v situaci, kdy je známa přesná poloha políčka na snímku, oblast políčka a jsou známy sousední políčka.

Dalším krokem je určení příslušné barvy políčka. Každá Rubikova kostka (klasické konstrukce) sestává z 6 stěn. Potom každá z 6 stěn má různou barvu, která je shodná pro 9 políček dané stěny. Pro bezpečné rozřazení vstupních informací jako jsou jednotlivé pixely příslušných políček do 7 (6) skupin lze využít metod shlukové analýzy. Knihovna OpenCV nabízí metodu `KMeans`.

Jedná se o iterativní algoritmus, který zdrojová data rozřazuje do shluků. Kritériem pro dělení je Euklidovská vzdálenost[5]. Výsledkem jsou středy jednotlivých shluků a každá vstupní hodnota je přiřazena právě do jednoho shluku. Pro dosažení lepších výsledků je tento algoritmus aplikován na data jednotlivých pixelů ve formátu LAB místo zdrojového RGBA. Hlavním důvodem je vzdálenost odlišných barev podle lidského vnímání, která je v tomto barevném modelu větší než u RGBA. Proto i výsledky při použití tohoto modelu vykazovaly větší úspěšnost.

[5] Vzdálenost vícerozměrných dat

Pro analýzu stěn Rubikovi kostky je vhodné rozřazení do 7 (6) shluků, 6 shluků uvažujeme takové hodnoty, kdy nedocházelo k zaznamenání černého okraje políčka. Výsledkem takové analýzy je rozdělení vstupních vzorků všech políček do právě 6 potřebných hodnot. Jako výsledná hodnota pro políčko je uvažován střed shluku, který obsahuje nejvíce pixelů z daného políčka. Zpětným přiřazením těchto středů odpovídajícím pozicím na Rubikově kostce dostáváme model kostky. Pokud takový model kostky projde procesem validace, je proces načtení úspěšně ukončený.

2.1.6 Testovaní algoritmu

Samotné dílčí kroky snímání jsou průběžně testovány na sadě testovacích dat. Testovací data sestávají z testovacích snímků zachycujících různé stavy kostky, různé světelné podmínky, kostky různých výrobců, a náhodné umístění stěny na snímků. Tyto snímky jsou doplněny o popisující soubor ve formátů JSON, ve kterém jsou popsány očekávané výsledky algoritmů.

Testy jsou navrženy tak, aby docházelo k automatickému spuštění konkrétního počtu testů. K tomu je využito funkce parametrických testů. Pro lepší práci a vývoj algoritmu rozpoznání jsou testy doplněny o třídu zpracovávající výsledky jednotlivých testů. Pro neúspěšné testy je po dokončení všech testů sestaven výsledek testů obsahující neúspěšné snímky spolu se snímky postupné detekce k odhalení příčiny selhání. Tyto výsledky jsou automaticky zaznamenány a tvoří data pro vyhodnocení úspěšnosti testu. V průběhu vývoje algoritmu byla sada testovacích dat rozšiřována v závislosti na zkoumaných podmínkách (okolní světelné podmínky, pozadí nebo pozice).

Hlavní testy rozpoznání byly členěny následovně:

- Testování existence kostky na snímku.
- Testování nalezení polohy jednotlivých políček.
- A testování celkového nasnímaní modelu.

Test existence kostky na snímku bylo nastaveno tak, aby testovaný snímek vyhověl testu, pokud obsahuje minimálně 9 obrysů. Pokud tomu tak bylo, test pokračoval druhou fází, a to testováním polohy.

Testování polohy bylo vyhodnoceno na základě určení středů jednotlivých políček, středy byly porovnány s očekávanou hodnotou v daném rozmezí. Pro vytvoření souboru s očekávanými

hodnotami byla využita jednoduchá aplikace autora, která slouží k vytváření souboru s očekávanými výsledky. Aplikace byla programována pomocí technologie JavaFX. Po načtení složky s testovanými obrázky byly jednotlivé obrázky popsány pomocí 9 bodů. Body bylo možné přetažením myši umístit na požadované místo na snímku (střed políčka). Následně bylo možné nastavit barvu políčka. Po popsání všech potřebných snímků byl vygenerován a uložen popisující soubor ve formátu JSON.

Tabulka 1: Výsledky testování rozpoznávacího algoritmu

	Počet snímků	Počet políček
Celkem	96	864
Úspěšné	80	720
Neúspěšné	16	144
Úspěšnost	83,333 %	

V tabulce lze pozorovat úspěšnost navrženého algoritmu přibližně s přesností 83 %. Takový výsledek je více než dostačující. Chyby byli nejčastěji sledovány u snímků, které byly zaznamenány v pohybu nebo ze špatného úhlu. K dosažení lepšího výsledku by bylo zapotřebí více výkonu. Takový algoritmus poté není vhodný pro použití v mobilním zařízení a nemuselo by se jednat o zpracování v reálném čase.

Test celkového načtení byl koncipován jako složení 6 odpovídající snímků jednoho stavu kostky. Výsledná podoba nasnímaného modelu byla testována proti očekávané podobě modelu. Tyto testy byly již navrženy v menším počtu pouze k ověření funkčnosti roztřídění jednotlivých nasnímaných barev do 6 skupin. Jinými slovy, pokud algoritmus našel stěnu nebylo limitující složit model kostky.

3 ŘEŠENÍ RUBIKOVY KOSTKY

Rubikova kostka je hlavolam, který lze prokazatelně složit do 20 tahů. Tah je myšleno otočení stěny o libovolný násobek úhlu 90°. Nalezení řešení na 20 tahů je celkem výpočetně náročné (kromě případů málo rozloženého stavu). Takové řešení nemá většinou logickou souvislost s jiným řešením. Proto je ze strany uživatel celkem nemožné se takový algoritmus naučit.

Z historie vzniklo mnoho technik skládání. Liší se zejména počtem, délkou a složitostí algoritmů[6]. Začátečníci volí metody s poměrně krátkými/jednoduchými algoritmy, kterých se stačí naučit omezený počet, a i tak vedou ke složení kostky. Použití těchto jednodušších algoritmů, ale znamená postupné řešení za použití více kroků a tím i prodloužení času nutného na řešení.

Pokročilí uživatelé využívají většího množství i složitějších algoritmů, které se skládají z většího počtu dílčích kroků v jednotlivých algoritmech. Využití těchto algoritmů vede zpravidla k nalezení řešení, která umožní realizaci řešení hlavolamu v kratším čase a za použití menšího počtu kroků.

3.1 Validace hlavolamu

Kostka je tvořena středovým křížem. Středový kříž pevně spojuje středy jednotlivých stěn, které jsou vzájemně vůči sobě vždy stejně umístěny. Ostatní dílky jsou pohyblivé, ale nikdy nelze ovlivnit pouze pozice a orientace jednoho dílku.

Aby byla Rubikova kostka složitelná, musí rozložení barev dodržovat daná pravidla. Od každé barvy smí být právě 9 políček. Na jednom dílku kostky nesmí sousedi protilehlé barvy středů.

Obecně platí, že pro složení kostky je nutný předpoklad správného mechanického fungování hlavolamu.

3.2 Metody skládání

Existuje tedy řada postupů, které může uživatel použít na složení Rubikovi kostky. Nejznámější jsou algoritmy “vrstva po vrstvě” (označována jako začátečnická metoda), těchto algoritmů využívá předchozí Bakalářská práce s průměrným počtem výsledných kroků 178. [1]

[6] Algoritmus se rozumí posloupnost elementárních tahů

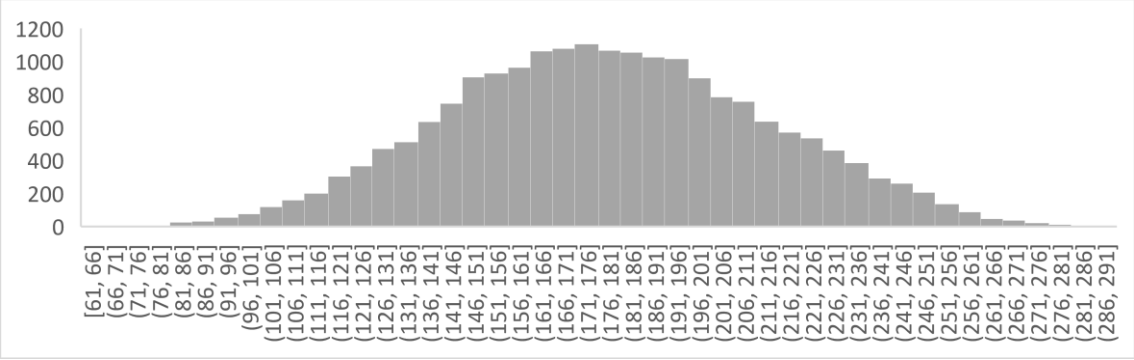
Jako vylepšená metoda je velice známa metod Fridrich (CFOP)[7]. Metoda Fridrich využívá více algoritmů s průměrným počtem výsledných kroků 52. Existuje mnoho různých variací na všechny algoritmy. Tyto variace vznikají především podle samotných uživatelů, kdy některé provedení kroků se jeví rychlejší nebo intuitivnější.

3.3 Navržené řešení

Výsledná aplikace implementuje pokročilou metodu Fridrich. Navíc je doplněna o schopnost nalezení lepšího řešení aplikací algoritmu na různě otočenou kostku. Výsledkem je 28 možných řešení, ze kterých je aplikace zvolí to nejkratší (tedy s nejmenším počtem nutných kroků).

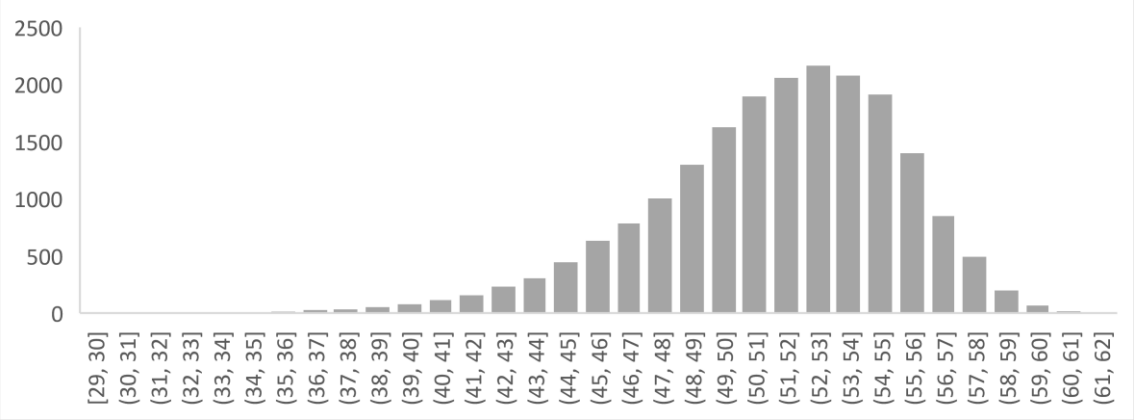
3.3.1 Výsledky řešení

Celá třída zodpovědná za hledání řešení je testována na 20000 generovaných počátečních stavech. Každý test hledá řešení počátečního stavu, které poté provede. Pro úspěšný test je nutné, aby řešení obsahovalo méně než 65 potřebných tahů a tyto tahy vedou uživatele až ke složení hlavolamu. Pro srovnání byly stejné počáteční stavy otestována na původní algoritmu Bakalářské práce. Výsledky testování obou aplikací lze pozorovat na grafech níže a také v porovnávací tabulce.



Obrázek 13: Graf četnostní počtů kroků původního algoritmu

[7] CFOP – Cross, F2L, OLL a PLL



Obrázek 14: Graf četností počtů kroků nového algoritmu

Tabulka 2: Statistické ukazatele testu algoritmů

	Původní algoritmus	Nové řešení
Minimální počet kroků	61	29
Maximální počet kroků	289	62
Průměrný počet kroků	178,6374	51,622
Střední počet kroků	178	52
Nejčastější počet kroků	169	53
Směrodatná odchylka	35,0705	3,981748

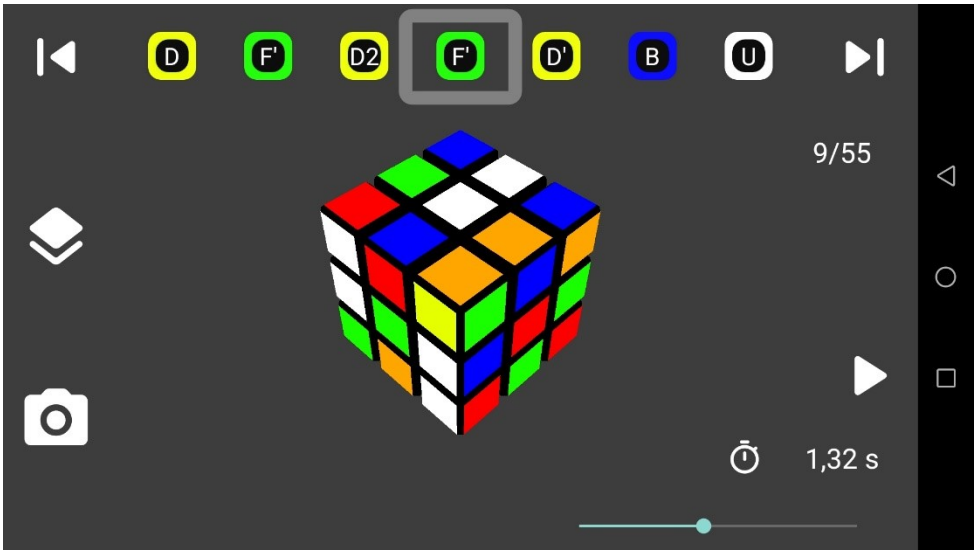
Z grafů a tabulky je patrné, že algoritmy jsou ve své úspěšnosti řádově odlišné. Minimální hodnota původního algoritmu je pouze o jeden krok menší než maximální hodnota nového řešení. Průměrný počet kroků u původního řešení je více než 3x větší, než je počet kroků u nového řešení. Původní řešení také vykazovalo celkem velký rozptyl výsledných hodnot daný odchylkou 35 kroků. Nové řešení poté pracuje s odchylkou pouze necelých 4 kroků.

4 VÝVOJ APLIKACE

Uživatelské rozhraní aplikace je navrženo jednoduše a srozumitelně. Uživatel ovládá celou aplikaci pomocí dvou uživatelských obrazovek. Obrazovky vyplňují celou plochu displeje. Z důvodu lepší ergonomie a uživatelského zážitku jsou obrazovky striktně orientovány na šířku. Úvodní obrazovka po zapnutí aplikace prezentuje načtený model s potřebnými kroky ke složení. Pomocí ovládacího prvku (ikona kamery) dojde v aplikaci k přesměrování na obrazovku načítání. Tato obrazovka zobrazuje snímky kamery v reálném čase a aktuální stav načítání pomocí vykresleného modelu.

4.1 Obrazovka s modelem

Obrazovka s modelem vykresluje model kostky. Po prvním zapnutí je připraven testovací model kostky pro seznámení uživatele s prezentací kroků a možnostmi ovládání modelu. Model lze na obrazovce rotovat pomocí dotyku (potažení) ve dvou osách. Při provádění pokynu rotace dochází na obrazovce k překreslení (rotaci) modelu v reálném čase. Reset pozice modelu do výchozího postavení lze provést pomocí ikony umístěné vlevo od modelu. Horní část obrazovky zobrazuje kroky nalezeného řešení. Šipkami lze ručně posouvat mezi jednotlivými pohyby. Každý pohyb je animován na základě nastavené doby animace (vpravo dole). K automatickému posunu slouží tlačítko spustit/pauza. Po spuštění jsou kroky automaticky posouvány po dokončení animování předchozího kroku.



Obrázek 15: Obrazovka modelu

Posuvník vpravo dole slouží pro nastavení doby animace jednoho kroku. Číslo 9/55 pod šipkou vpravo znázorňuje aktuální pořadí kroku/celkový počet kroků.

4.1.1 Model kostky

Pro vykreslení pokročilé grafiky není vhodné použít již připravené komponenty, které nabízí standartní aplikace. Lepších vizuálních efektů lze docílit vlastní implementací grafického vykreslování. K tomuto účelu je vytvořeno mnoho grafických knihoven. Jedny z mnoha jsou Unity, Unreal Engine a Fusion.

Tyto nástroje slouží pro rychlé vytváření složitých grafických scén. Knihovny zabezpečují jednak vlastní vykreslování, ale třeba také interakci s vykreslenými objekty. Pro implementaci v tomto projektu by použití takové knihovny mohlo být kontraproduktivní. Zejména z pohledu časové náročnosti studovaní možnosti a práce s knihovnou. Pro jednoduchou vizualizaci 27 stejných kostek lze efektivně použít základní grafické API OpenGl.

Model kostky je vykreslován přesněji pomocí OpenGL ES API. Pro vývoj se používají jednotlivé vrcholy v 3D prostoru spojované po 3, a tím tvořící trojúhelník. Každý vrchol může být definován barvou, kterou je poté vykreslen odpovídající trojúhelník. Druhou možností je nanesení texturové informace předané pomocí předlohy například fotografie. Pomocí trojúhelníku vhodné velikosti a barvy případně textury lze modelovat jakýkoliv tvar. Využitím většího množství trojúhelníků lze dosáhnout přesnějšího detailnějšího modelu. Nevýhodou je, že k vykreslení takového modelu je potřebný větší výkon použitého HW. Proto je třeba nalézt optimální poměr mezi kvalitou modelu a potřebným výkonem k jeho stvoření. [5]

Další využívanou technikou je transformace vrcholů. Transformace je prováděna pomocí transformační matice. Aplikací transformace lze přepočítat pozice odpovídajících vrcholů. Transformaci lze využít k otáčení objektů, posunů ve scéně, deformaci nebo ke změně velikosti objektu. [5]

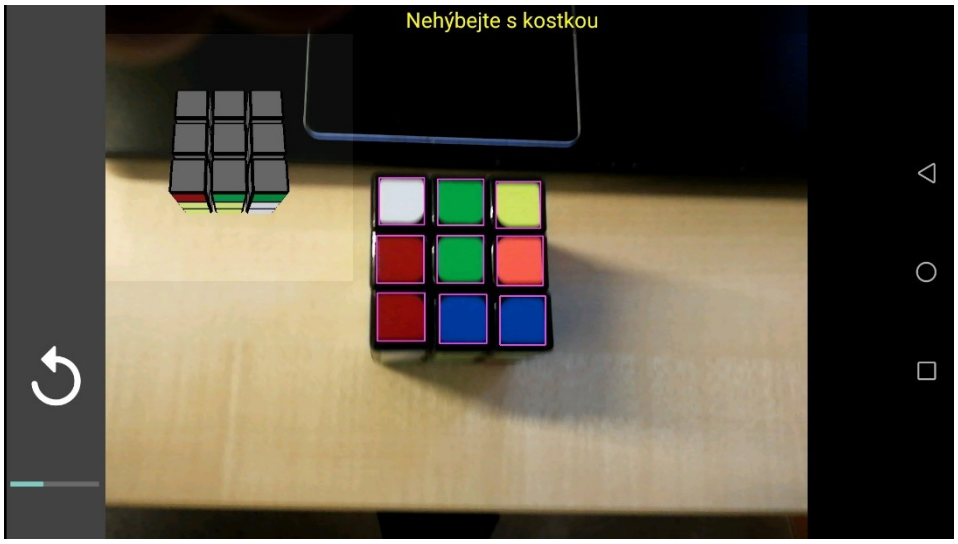
Pro simulaci pohledu kamery je využito další transformační matice. Jedná se tedy o změnu celé scény, která se jeví pro uživatele jako pohyb kamery. [5]

Pro věrnou reprezentaci kostky je model detailně vykreslen jako kostka, která je složena z dílčích kostek. Každá dílčí kostka je vyobrazena jako kostka se zkosenými hranami. Animace jsou vykreslovány pomocí přibližně 50 snímku za sekundu. Při každém snímku dojde k odpovídající změně v transformaci modelu. Po dokončení animace jsou všechny transformace vymazány a model je překreslen na základě uloženého globálního modelu kostky. Vzhledem k jednotným

barvám jednotlivých políček není využíváno texturování, ale je použita jednotná barva pro celé políčko.

4.2 Obrazovka načítání

Model do aplikace lze zadat pouze pomocí načtením všech stěn kostky. Načtení je prováděno v daném pořadí stěn. Uživatel je o potřebě otočit kostku informován informační lištou v horní části obrazovky. Vlevo je aktuální proces načítání zobrazován opět na 3D modelu. Tento model zároveň slouží pro kontrolu již zaznamenaných informaci, tak pro animaci potřebné rotace kostky pro další načtení. Pro opakování načtení v probíhajícím procesu lze použít tlačítko reset. Po úspěšném načtení modelu kostky je uživatel automaticky přesměrován na obrazovku modelu. Zároveň je nalezeno řešení právě nasnímaného modelu. Model navíc prezentuje barvy a barevné tóny políček reálné kostky. Při neúspěšném načtení (selže validace modelu) je uživatel informován zprávou a vyzván k opakování procesu načtení.



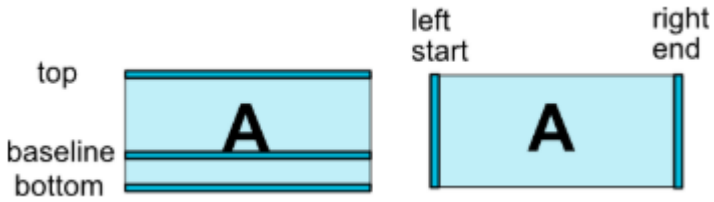
Obrázek 16: Obrazovka načítání

Načtení jedné stěny je provedeno po pěti snímcích, na kterých byla detekována stěna kostky. Tento průběh je znázorněn na obrazovce pomocí indikátoru průběhu vlevo dole. Poslední snímek je následně zaznamenán a slouží pro sestavení modelu.

Z důvodu snížení výkonnostních nároků není prováděna detekce v okamžiku, kdy je rotován model, v této situaci se předpokládá od uživatele samotné otočení kostky v požadovaném směru.

4.3 Grafické rozhraní

Grafické rozhraní aplikace je vytvořeno jako responzivní dizajn. Pro navržení takového dizajnu je využita knihovna `ConstrainLayout`. Pozice prvků na displeji se definuje za pomocí takzvaných omezení. Prvek se umístí relativně na základě pozice rodiče, nebo jiných definovaných ostatních prvků. Využívá se určování pozice v závislosti na 5 definovaných místech na každém prvku znázorněných na obrázku níže. [6]

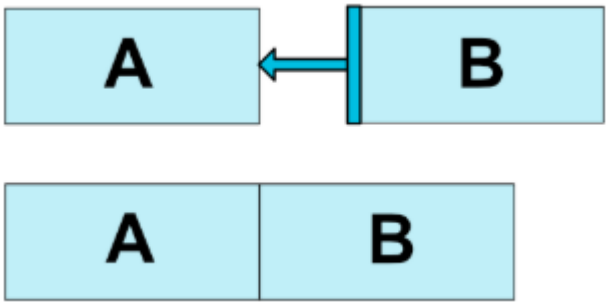


Obrázek 17: Omezení relativní pozice převzato z developer.android.com

Zápis omezení je poté možný například jako:

```
<Button android:id="@+id/buttonA" ... />
<Button android:id="@+id/buttonB" ...
app:layout_constraintLeft_toRightOf="@+id/buttonA" />. [6]
```

Výsledkem je pozice tlačítek za sebou jako na obrázku níže.



Obrázek 18: Ukázka funkce omezení převzato z developer.android.com

5 DATOVÉ STRUKTURY A ALGORITMY

Následující struktury a algoritmy pro přehlednost budou rozděleny na část zpracování obrazu, část modelu kostky a ostatní části.

5.1 Část modelu kostky

Stav aplikace je uložen za pomoci návrhového vzoru jedináčka. Jedináček `GlobalState` uchovává jednu instanci kontroléru, nastavené hodnoty pro čas animace a třídu určenou pro uchování stavu načítání `CubeScan`. Propojení tříd lze sledovat na diagramu níže, podrobný je také zařazen v příloze A.

Obrázek 19: Diagram hlavních tříd

Model kostky je uložen pomocí třídy `CubeModel`. Tato třída nese informace o vzájemné poloze políček, a také o barvách vykreslování.

Pro pohybování modelu slouží třída `CubeModelController`. Tato třída nabízí rozhraní pro provádění kroků na modelu a zároveň pro nalezení řešení. Vývoj těchto tříd byl prováděn zejména proti jednotkovým testům pro ověření správné funkčnosti kritických částí kódu.

5.1.1 Třída `CubeModel`

Třída `CubeModel` uchovává kompletní podobu modelu v paměti. Model je reprezentován jako 6 vzájemně propojených stěn kostky, které jsou reprezentovány pomocí třídy `CubeLayer`. Samotné stěny uchovávají reference na nejbližší sousední stěny a matici reprezentující rozložení barev na stěně. Rotace je prováděna přesunem barev v matici a zároveň přesunem barev mezi sousedními stěnami. Třída dále uchovává hodnoty reálně načtených barev ve formátu RGBA. Tyto hodnoty slouží při vykreslování modelu k věrné prezentaci barev reálné kostky. Třída pro zamezení chyb a vytváření nevalidních modelů ve svých konstruktorech provádí validaci modelu a při neúspěchu vytváří výjimku.

5.1.2 Třída `CubeModelController`

Třída `CubeModelController` zabezpečuje funkci kontroléru celého model. Zejména zpřístupňuje možnost provádět kroky a nalézat řešení. Provádění kroků je řešené pomocí příkazů předávaných jako textový řetězec. Příkazy lze předat jednotlivě nebo jako sadu oddělené čárkou. Možné příkazy jsou převzaty z kapitoly 1.1.2 Princip hlavolamu Bakalářské práce. [1]

5.1.3 Třída `CubeUtils`

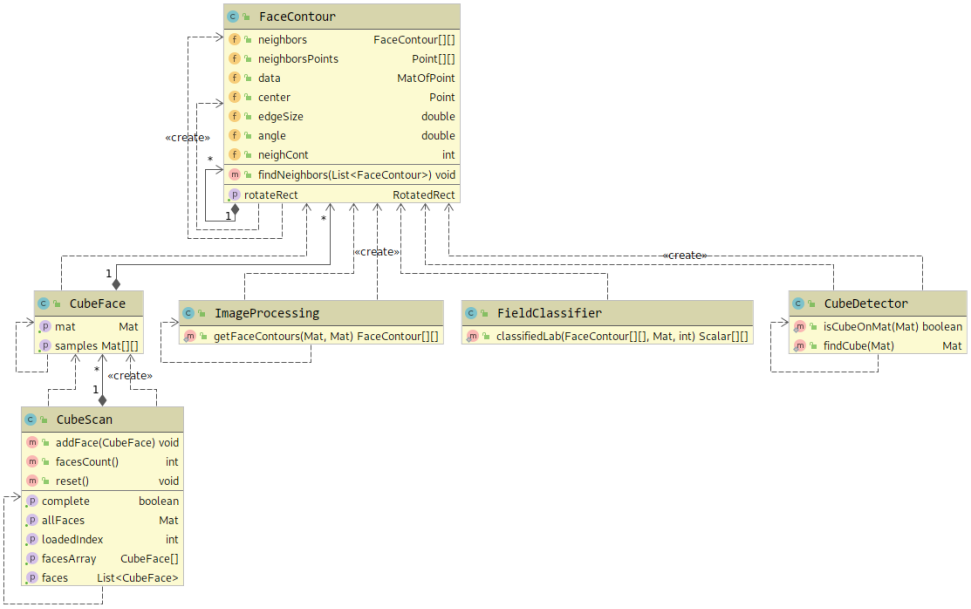
Pomocná třída `CubeUtils` je využívaná k hledání jednotlivých kostiček v modelu, vracení aktuálních poloh nebo barev políček. Dále je rozšířena o funkce minimalizující počet kroků, které odstraňují redundantní nebo duplicitní kroky v sadě příkazů.

5.1.4 Třída `CubeLayer`

Třída `CubeLayer` nesoucí pole barev jedné stěny kostky. Dále využívá reference na sousední stěny pro zajištění možnosti otáčení s celou vrstvou. Samotné otočení je provedeno pomocí přeorganizování pole barev a přenesením správných částí polí mezi sousedními stěnami.

5.2 Část zpracování obrazu

Tato část popisuje způsob navržení a implementace struktur potřebných pro zpracování obrazu z kamery. Diagram níže ukazuje vzájemné propojení popisovaných tříd.



Obrázek 20: Diagram tříd zpracování obrazu

5.2.1 Třída `FaceCotour`

Třída `FaceCotour` reprezentující jedno políčku načítaného snímku. Hlavní funkce je vytváření návazností mezi jednotlivými políčky načtené stěny. To znamená detekci, na jaké pozici se políčko nachází ve vztahu k ostatním. Z veškerých filtrovaných obrysů jsou vytvořeny instance této třídy. Následuje testování vzájemných poloh. Pokud se v předpokládané oblasti nachází jiný obrys je nastaven jako soused na dané pozici. Výsledkem je potom takový obrys, který nalezl veškeré své sousedy a takový je nastaven jako střední obrys. Z tohoto obrysu je zpětně sestavena instance třídy `CubeFace`.

5.2.2 Třída `CubeFace`

Třída `CubeFace` reprezentuje načtenou jednu stěnu z jednoho snímku. Je složena z matice jednotlivých vzorků políček (výřezy z originálního snímku) a originální snímek.

5.2.3 Třída `CubeScan`

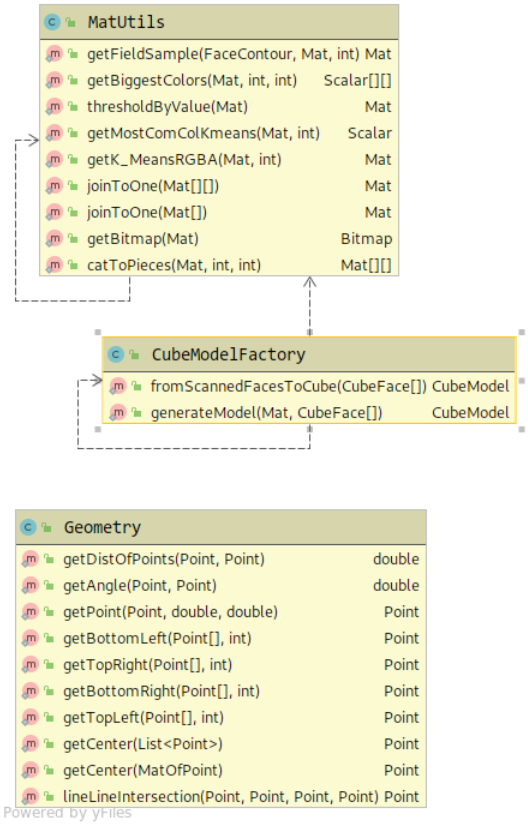
Třída `CubeScan` slouží k uchování stavu načítaní celého modelu. Drží 6 instancí jednotlivých `CubeFace`. Po načtení všech 6 stěn je vytvořen model kostky, který je následně uložen do jedináčka aplikace (pokud se jedná o validní model).

5.3 Ostatní části

V této části budou popsány ostatní funkce aplikace zejména sloužící pro vytváření uživatelského rozhraní a dalších funkcí.

5.3.1 Pomocné třídy

Aplikace využívá pomocné třídy zejména pro výpočty geometrických hodnot a pro úpravu matic snímků.



Obrázek 21: Diagram pomocných tříd

Třída `MatUtils` shromažďuje funkce pro práci s maticemi reprezentující snímky. Jednak jsou zde zařazeny funkce pro vyřezávání jednotlivých políček ze zdrojového snímku, poté funkce určování a klasifikaci barev do již zmiňovaných skupin.

Třída `CubeModelFactory` je pro transformaci ze skenovaných dat do modelu kostky. Třída spojením jednotlivých výřezů políček v daném pořadí a aplikaci zmiňované funkce `KMeans` rozdělí políčka na 6 skupin a zpětně se pokusí o vytvoření instance třídy `CubeModel`. Pokud došlo k vytvoření instance modelu je následně vrácena nebo je vystavena výjimka a proces načítání je nutné opakovat.

6 MOŽNÁ ROZŠÍŘENÍ A VYLEPŠENÍ

U problematiky rozpoznání kostky v obraze by bylo možné řešení rozšířit o načítání více stěn najednou. Toto řešení by kompletně odstranilo nutnost načítání stěn kostky v daném pořadí. Pravděpodobně by ale takto navržená aplikace nebyla schopna pracovat v reálném čase na mobilním zařízení z důvodu nedostatečného výpočetního výkonu cílových zařízení. Méně náročnou možností by mohla být analýza více snímků jedné stěny, kde by se přesnost načítání zvětšila především zpracováním více informací o jedné stěně.

Načítání by bylo také přehlednější využitím rozšířené reality, kdyby se přímo potřebné kroky promítaly do reálného světa, tedy přímo na reálnou kostku. Tohoto řešení by bylo možné využít i pro prezentaci řešení pomocí vykreslení potřební kroků přímo na reálnou kostku.

Hledání potřebných kroků řešení by bylo možné optimalizovat využitím například umělé inteligence nebo použitím algoritmu, který by hledal řešení pomocí matematických permutací nebo optimalizoval již nalezené řešení. Tento algoritmus by ovšem nebylo možné doporučit jako skládací techniku pro uživatel, který má za cíl naučit se skládat Rubikovu kostku.

Aplikaci bylo možné rozšířit zejména z pohledu jednoduchosti obsluhy, a to doplněním tutoriálu spolu s nápovědou. Také by bylo vhodné přidat vysvětlení jakým způsobem aplikace hledá řešení především jakou techniku skládání využívá.

Pro rozšíření uživatelské základny by bylo vhodné aplikaci přeložit i do více jazyků například angličtina, němčina případně čínština.

7 VÝVOJOVÉ PROSTŘEDÍ A TECHNOLOGIE

Celá aplikace byla vyvíjena na operačním systému Linux za pomoci vývojového prostředí Android Studio od Společnosti Google. Pro správu verzí programu bylo využito verzovacího nástroje GIT. Záloha aplikace byla udržována na vzdáleném repositáři Bitbucket.org od společnosti Atlassian. Aplikace je vyvinuta pro minimální verzi operačního systému Android 5.0. Pro instalaci je využito jednoho souboru typu apk. Vývoj a testování probíhalo na reálném zařízení Huawei Mate 8 s operačním systémem ve verzi 8.0.0. Výsledná aplikace poté testována na zařízení Honor 8X, Lenovo Moto E4, Xiaomi Mi A2 Lite, Xiaomi Redmi Note 4 a Huawei MediaPad T3 7.

ZÁVĚR

Hlavním cílem práce byla inovace stávající aplikace popsané v Bakalářské práci hned v několika ohledech. Veškeré problémy se podařilo potlačit případně odstranit.

Pro rozpoznání stěny kostky byl navržen takový algoritmus, který pracuje v nepoměrně širším spektru možných světelných podmínek. Od přímého slunečního svitu až po šero, nebo dokonce tmu s využitím přisvětlovací diody zařízení. Je také odolný vůči umělému osvětlení například světlem ze žárovky, které mělo snad nejzásadnější podíl na nepřesném určování hodnoty barev na stěně kostky. Jediná slabina takového algoritmu je v odlescích barevných nálepek, které na snímcích zakrývají okraje políček, a proto není možná následná detekce. Tento jev je jednoduché eliminovat změnou úhlu natočení stěny tak, aby nedocházelo k odleskům.

Díky tomu, že nový algoritmus zpracovává všech 6 stěn kostky současně, lze načítat jakoukoliv kostku z pohledu její barevné palety. Navíc se reálná paleta věrně převede do reprezentovaného modelu v aplikace. To pomáhá zejména pozorovat potřebné kroky a sledovat aktuální podobu kostky v průběhu skládání. Pokud by uživatel provedl špatný krok, je ihned v aplikaci vizuálně znázorněn rozdíl modelu kostky a kostky v reálném světě.

Editace nasnímaného modelu byla v nové verzi aplikace odebrána úplně. Kostku do aplikace lze zaznamenat pouze pomocí kamery mobilního zařízení. Aplikace neumožní ukončení načítacího procesu v případě, že model kostky není validní. Navíc lze načíst kostku, která nelze složit (např. otočen jeden roh) pouze algoritmus hledání nenalezne správné řešení.

Nový algoritmus skládání je v porovnání s algoritmem z Bakalářské práce výrazně úspěšnější. Výsledné hodnoty měření jsou téměř nesrovnatelné s původní verzí. Z testovaných 20000 generovaných počátečních řešení byly hodnoty nejmenšího počtu původního algoritmu pouze o 1 krok menší, jak maximální počet kroků nového algoritmu. Tomu odpovídají i naměřené hodnoty středních hodnot 178 u původního algoritmu a 52 u nového algoritmu.

Také z pohledu kvality bylo na testech prokázáno, že nový algoritmus vykazuje konzistentnější výsledky s odchylkou pouze necelých 4 kroků. Tokový to algoritmus je mnohem přívětivější pro uživatele, který díky tomu není nucen opakovat více než 150 kroků, aby se dostal k tíženému výsledku.

Ve srovnání s předchozí verzí je celý proces práce s aplikací razantně zkrácen. Načtení je prováděno rychleji zejména pomocí odstranění editace načtené kostky. Načítání v přibližně 90 % případů lze ukončit na první pokus. Výsledný počet nutných kroků je nejméně 3x menší než

u původní aplikace. Výsledkem je tedy výrazné zkrácení doby, kterou uživatel potřebuje od rozložené kostky k úspěšnému složení kostky.

POUŽITÁ LITERATURA

[1] *Rozpoznání a složení Rubikovy kostky* [online]. Pardubice, 2017 [cit. 2019-04-17]. Dostupné z: https://dk.upce.cz/bitstream/handle/10195/68556/MachovecL_RozpoznaniSlozeni_KS_2017.pdf?sequence=1&isAllowed=y. Bakalářská práce. Univerzita Pardubice, Fakulta elektrotechniky a informatiky. Vedoucí práce Šimerda, Karel.

[2] About – OpenCV library. *OpenCV library* [online]. 2019 [cit. 2019-04-13]. Dostupné z: https://opencv.org/about.html

[3] OpenCV 3.4.3 Java documentation. *OpenCV documentation index* [online]. 2018 [cit. 2019-04-13]. Dostupné z: https://docs.opencv.org/3.4.3/javadoc/index.html

[4] CANNY, John. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1986, **PAMI-8**(6), 679-698. DOI: 10.1109/TPAMI.1986.4767851. ISSN 0162-8828. Dostupné také z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4767851

[5] *LearnOpenGL: Your #1 resource for OpenGL* [online]. Nizozemsko: Joey de Vries [cit. 2019-04-19]. Dostupné z: https://learnopengl.com/

[6] Build a Responsive UI with ConstraintLayout. *Android Developers* [online]. [cit. 2019-05-04]. Dostupné z: https://developer.android.com/training/constraint-layout

PŘÍLOHY

PŘÍLOHA A – PODROBNÝ DIAGRAM MODELU KOSTKY