VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

BAKALÁŘSKÁ PRÁCE

2017 Štěpán Němec

VŠB – Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra kybernetiky a biomedicínského inženýrství

Systém pro lokalizaci předmětu využívající obrazový signál

System for Localization based on Image Signal Analysis

2017 Štěpán Němec

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra kybernetiky a biomedicínského inženýrství

Zadání bakalářské práce

Student: **Štěpán Němec**

Studijní program: B2649 Elektrotechnika

Studijní obor: 2612R041 Řídicí a informační systémy

Téma: Systém pro lokalizaci předmětu využívající obrazový signál
System for Localization based on Image Signal Analysis

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Rozbor problematiky zpracování obrazového signálu.
2. Návrh systému pro lokalizaci předmětů s využitím obrazového signálu.
3. Realizace systému pro lokalizaci a vyhodnocení pozice předmětů s využitím obrazového signálu.
4. Verifikace systému pro lokalizaci s vyhodnocením skutečných parametrů systému.
5. Zhodnocení dosažených výsledků.

Seznam doporučené odborné literatury:

[1] DOBEŠ, Michal. *Zpracování obrazu a algoritmy v C#: automatické rozpoznávání, úprava snímků, příklady zdrojového kódu.* 1. vyd. Praha: BEN - technická literatura, 2008. 143 s. ISBN 978-80-7300-233-6.
[2] SOJKA, Eduard. *Digitální zpracování a analýza obrazů.* 1. vyd. Ostrava: VŠB-Technická univerzita Ostrava, 2000. 133 s. ISBN 80-7078-746-5.
[3] SOJKA, Eduard. *Matematické základy počítačové grafiky ; Matematické základy digitálního zpracování obrazu* [CD-ROM]. Ostrava: VŠB - Technická univerzita Ostrava, 2012 [cit. 2016-11-01].

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Zdeněk Macháček, Ph.D.**

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017

VYSOKÁ ŠKOLA BÁŇSKÁ - TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * 1D16 * OSTRAVA *

doc. Ing. Jiří Koziorek, Ph.D.
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 28. dubna 2017 .....

Rád bych touto cestou poděkoval vedoucímu bakalářské práce **Ing. Zdeňku Macháčkovi, Ph.D.** za odbornou pomoc a cenné rady při tvorbě této práce.

Abstrakt

Tato práce se zabývá zpracováním obrazu vedoucímu k úspěšné lokalizaci jakéhokoliv předmětu. Sejmutý obraz se kvůli libovolným vnějším vlivům může potýkat se spoustou nedokonalostí, a tak je zde uvedeno studium veškerých metod, kterými je možné tyto vnější vlivy odstranit a úspěšně lokalizovat hledané předměty. Jsou zde popsány hlavně metody filtrace obrazu, geometrické transformace, hranové detekce, segmentace a matematické morfologie. Cílem práce je z nastudovaných poznatků navrhnout a naprogramovat lokalizační systém, který bude schopen na základě obrazového signálu snímaného kamerou lokalizovat jednotlivé předměty logické hry s názvem Hanojské věže. Na základě dat ze zpracování bude tento systém schopen analyzovat provedené kroky řešitele, a tak může plnit funkci kamerového rozhodčího.

Klíčová slova: zpracování obrazu, detekce hran, lokalizace předmětů

Abstract

This work deals with image processing leading to the successful localization of any subject. The scanned image can be faced with many imperfections due to any external influences, so there is a study of all the methods by which these external influences can be removed and sought objects can be succesfully localized. There are described mainly the methods of image filtration, geometric transformation, edge detection, segmentation and mathematical morphology. The aim of the thesis is to design and program a localization system, which will be able to locate the individual objects of the logic game named Hanoi Tower based on the video signal captured by the camera. Based on the data from the processing, the system will be able to analyze the steps taken by the investigator, and thus perform the function of a camera judge.

Keywords: image processing, edge detection, objects localization

Obsah

Seznam použitých symbolů a zkratek

$g\,(x, y)$	bod výstupního obrazu
$f\,(x, y)$	bod vstupního obrazu
$T\,[\,]$	operace s proměnnou v závorce
$h\,(s, t)$	konvoluční matice
$f\,(x–s, y–t)$	konvolucí zpracovávaný vstupní obraz
$h,\ h_1, h_2, h_3$	konvoluční matice operátoru

Seznam obrázků

Seznam tabulek

1. Úvod

Ze snímané scény, která je zpracována vhodným algoritmem, je možné zjistit velké množství informací, jako jsou například informace o lokaci libovolného objektu, jeho rozměru, množství, barvě a mnoho dalších. Právě díky takto velké škále informací nacházejí úlohy se zpracováním obrazu uplatnění v různých odvětvích a oborech. Můžou to být například jednoduché aplikace pro různé typy her. Nebo ty složitější, kterými mohou být aplikace pro rozpoznání obličeje, aplikace vyskytující se v průmyslové automatizaci pro rozpoznání vadných kusů, správného pohybu robotů, počítání výrobků atd.

Vzhledem k různým vlivům nepříznivě působícím na sejmutý obraz se může ve zpracovávaném obraze vyskytovat některý z typů šumu. Možnosti k jejich odstranění a zlepšení obrazu k dalším pracím jsou uvedeny v kapitole 2. V kapitole 3 jsou poté popsány postupy a operátory pro detekci hran, segmentaci obrazu pomocí prahování a také operace matematické morfologie, kterými se často zdokonaluje obraz pro identifikaci objektů.

Veškeré nastudované poznatky jsou poté využity při návrhu a realizaci kamerového rozhodčího pro logickou hru s názvem Hanojské věže. Jelikož z velkého množství metod pro jednotlivé kroky při zpracování obrazu není možné jednoznačně určit, jaký postup zvolit a které metody využít, je v kapitole 4 rozebráno, jak jednotlivé metody působí na zpracovávaný obraz Hanojských věží. Z tohoto rozboru je poté navrhnut systém, který se jevil jako nejspolehlivější. Na konci této kapitoly jsou ještě uvedeny algoritmy pro identifikaci jednotlivých objektů Hanojských věží ze zpracovaného obrazu. Kapitola 5 pak už pojednává o finální aplikaci grafického rozhraní a je hlavně věnována rozhodovacím algoritmům, které se starají o logické uvažování naprogramovaného kamerového rozhodčího.

2. Filtrace obrazu

Ne vždy zachytí kamera pořizovaný obraz tak, jak je očekáváno, jelikož na ní působí několik možných faktorů, jako je například špatná expozice, protisvětlo, velké kontrasty stínu, nebo světlé části scény. Pomocí několika metod filtrace obrazu, které budou popsány v této kapitole, je snaha tyto faktory v co největší míře eliminovat tak, aby nakonec výsledný obraz vypadal přesně podle očekávání.

Veškeré operace s obrazem jsou prováděny buď v *prostorové doméně*, nebo ve *frekvenční doméně*. Pojem zpracování v *prostorové doméně* znamená, že vstupní obraz je představován jako dvourozměrná matice, která má rozměry podle rozměru obrazu a jednotlivé hodnoty jsou odpovídající hodnotám jasu daného pixelu černobílého obrazu, který se zde nachází. Na tuto matici se poté při zpracování provádí jednotlivé metody, které zde budou popsány. [1, s. 31] Základní princip zpracování ve *frekvenční doméně* je založen na převedení vstupního obrazu lineární integrální transformací (např. Fourierovou transformací) do frekvenční domény, kde se obraz vyfiltruje a potom se inverzní lineární integrální transformací převede zpět do časové domény. [4, s. 39]

Tato bakalářská práce se bude zabývat z větší části *prostorovým* neboli také *bodovým* zpracováním obrazu. Základní vztah mezi vstupním a výstupním obrazem je vyjádřen v rovnici (1).

$$g(x, y) = T[f(x, y)] \tag{1}$$

U nejjednodušší operací jsou vyžadovány pouze znalosti právě jednoho bodu, se kterým se zrovna pracuje, zatímco složitější operace mohou potřebovat například maximální, nebo minimální hodnotu jasu v obraze, histogram apod. U těch nejsložitějších operací se navíc pracuje i s okolními body pomocí tzv. konvoluční masky. Jak už bylo zmíněno, operace se provádí s hodnotami jasu černobílého obrazu, kde velikost jasové stupnice bývá určena počtem bitů paměti jednoho pixelu a tento počet je nutno správně zvolit. Nejideálnější počet bitů, se kterým se pracuje pro běžné zpracování, se rovná osmi, což je 256 jasových úrovní. Při potřebě kvalitnější rekonstrukce signálu lze ovšem použít i větší počet bitů. [1, s. 31-32]

2.1. Geometrické transformace

Při zpracování obrazu se může stát, že je potřeba vstupní obraz geometricky upravit. Mezi takové základní úpravy patří posunutí, otočení, změna velikosti, zkosení apod. [1, s. 49] Tyto transformace jsou velice důležité v případě potřeby eliminovat geometrické zkreslení vzniklé kamerou při pořizování obrazu. *Geometrická transformace* jako celek se skládá ze dvou kroků. Prvním krokem je tzv. *transformace souřadnic bodů* a druhým krokem je *aproximace jasové funkce*. [4, s. 30-31]

2.1.1. Transformace souřadnic bodů

Transformace souřadnic se pro názornost a menší složitost počítají a vyjadřují v homogenních souřadnicích, kde se dá použít maticový zápis. V této kapitole budou uvedeny rovnice pro výpočet základních operací, kde vektor $\bar{a} = [a_x \quad a_y \quad 1]$ je vektor se souřadnicemi, které budou transformovány a vektor $\bar{b} = [b_x \quad b_y \quad 1]$ je vektor s výslednými souřadnicemi, kde se bude nacházet počítaný bod. Jak lze vidět, tak u každého vektoru je jako 3. složka přidána 1, aby bylo

možné počítat nové souřadnice pomocí matice. Rovnice (2) vyjadřuje posunutí bodu o t_x ve směru x a o t_y ve směru y. Rovnice (3) pak vyjadřuje otočení okolo počátku o úhel φ a rovnice (4) znázorňuje změnu měřítka s_x ve směru x, resp. s_y ve směru y.

$$[b_x \quad b_y \quad 1] = [a_x \quad a_y \quad 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} \tag{2}$$

$$[b_x \quad b_y \quad 1] = [a_x \quad a_y \quad 1] \begin{bmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

$$[b_x \quad b_y \quad 1] = [a_x \quad a_y \quad 1] \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

Pokud by bylo potřebné provést otočení kolem jiného bodu než kolem počátku souřadnic, tak se provede *složená transformace*, která provede posunutí bodu, od kterého je chtěno provést rotaci, do počátku, dále se provede otočení a nakonec posunutí zpět. Pokud je potřebné provést s obrazem více operací naráz, tak je možné jednotlivé transformační matice jednoduše mezi sebou vynásobit a výslednou maticí pak vynásobit vstupní vektor. [1, s. 49-50]

2.1.2. Aproximace jasové funkce

Po transformaci souřadnic bodů následuje druhá úloha *geometrické transformace*, která spočívá v nalezení správné hodnoty jasu pro daný pravoúhlý rastr tvořen maticí s x řádky a y sloupci podobně jako vstupní obraz. Jinak řečeno souřadnice vstupního obrazu jsou celočíselné hodnoty a_x a a_y, ovšem u vypočítaných výstupních souřadnic b_x a b_y tohle neplatí, a tak úlohou *aproximace* je nalézt odpovídací jasové hodnoty v celočíselných souřadnicích co nejpřesněji.

Nejjednodušší metodou je *metoda nejbližšího souseda*, která spočívá v tom, že požadovanému finálnímu bodu v rastrové mřížce je přiřazena jasová hodnota nejbližšího vypočítaného bodu b v diskrétní mřížce. [4, s. 32-33] Jelikož výsledky této metody jsou nepříliš uspokojivé z důvodu ztráty detailů, tak v praxi nenachází příliš velké uplatnění. [1, s. 51] Chyba této metody může dosahovat až půl pixelu.

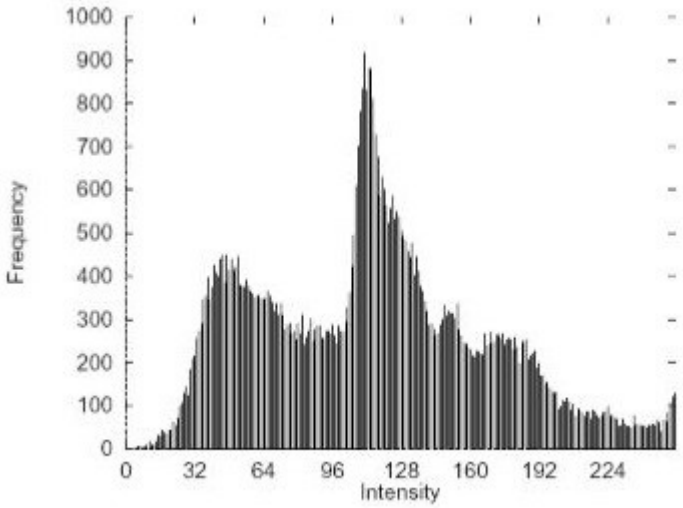
Z tohoto důvodu je více oblíbená *metoda lineární interpolace*, u které je pro výpočet výsledného jasu zpracovávaného bodu rastrové mřížky využívána jednak samotná hodnota jasu, ale také i poloha okolních nejbližších čtyř bodů. Přitom vliv každého okolního bodu odpovídá jeho vzdálenosti od zpracovávaného bodu. *Lineární interpolace* lehce nepříznivě ovlivňuje rozlišení, jelikož při průměrování okolních bodů je obraz vyhlazován lineárním filtrem, a tak může docházet k malému rozmazávání. Toto rozmazávání je ovšem o mnoho menší zlo než schodovitost, která se projevuje u *metody nejbližšího souseda*. Poslední metodou aproximace jasové funkce je tzv. *bikubická interpolace*, u které je využito okolních šestnácti bodů, tudíž má nejlepší vlastnosti co se týče zachování detailů a minimálnímu rozmazávání, takže je tedy nejvíce používaná v oblasti rastrových displejů a zobrazovacích programů. [4, s. 33-34]

2.2. Jasové transformace

Snímací a digitalizační zařízení by mělo mít v ideálním případě stejnou citlivost na všechny body v obraze, avšak v praxi tento požadavek není většinou splněn. U optických soustav bývá světlo, které prochází déle od optické osy zeslabováno, tomuto jevu se říká vinětace. Dalším problémem může být například CCD prvek, který není úplně nejkvalitnější, a tak není stejně citlivý ve všech bodech. Zdrojem poruch by mohla být také špatně osvětlená snímaná scéna, nebo prachové částice na skle optického snímače. Pokud jsou tyto poruchy systematické, což znamená že nejsou náhodné, dají se jednoduše pomocí vhodně zvolené funkce, která bude fungovat po celou dobu snímání stejně, eliminovat. Pokud ovšem jsou náhodné, tak je potřeba k eliminaci využít metody *lokální filtrace*, nebo *lokálního vyhlazování*. [4, s. 35]

2.2.1. Histogram

Histogram je sloupcový graf, který udává informaci o výskytu jednotlivých jasových úrovní v obraze. Na ose *x* se vyskytují úrovně jasové stupnice a na ose *y* je pak zobrazen jejich počet. Informace z *histogramu* se často využívají ke zlepšení vzhledu (viz kapitola 2.2.2), nebo také k prahování (viz kapitola 3.2). [1, s. 40] *Histogram* bývá většinou jedinou globální informaci o obraze a jelikož udává pouze informaci o počtu jednotlivých jasových hodnot, tak pomocí něj není možné zjistit například pohyb objektu na snímané scéně, jelikož při tomto pohybu bude *histogram* nadále stejný. Ukázka takového *histogramu* je vidět na *Obr. 1.* [5, s. 29-30]

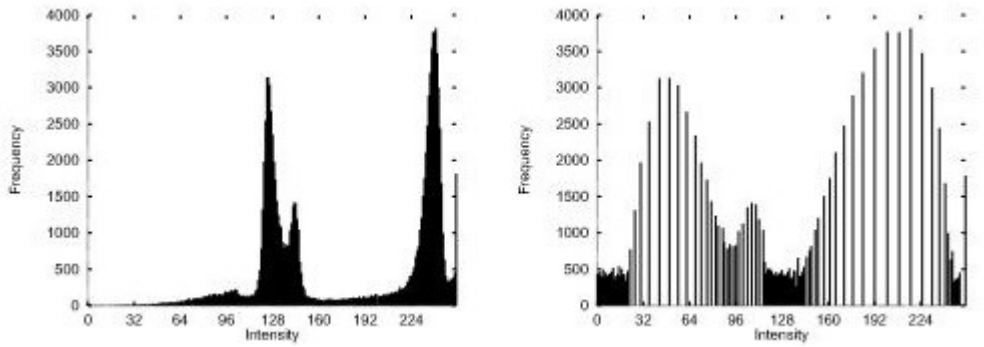


Obr. 1 Ukázka histogramu

2.2.2. Ekvalizace histogramu

Ekvalizace neboli *vyrovnání histogramu* představuje operaci, kterou se docílí rovnoměrného rozložení hustoty jasu výsledného obrazu. Toto *vyrovnání* s velkou pravděpodobností způsobí jednak zlepšení kvality obrazu vnímané pozorovatelem, ale také i zlepšení podmínek pro další zpracování. Ovšem mohou nastat i případy, u kterých *ekvalizace* bude mít negativní vliv na zpracovávaný obraz. [3, s. 73] Na *Obr. 2* vlevo je vidět *histogram* původního obrazu, kde jsou nejčetnější jasové úrovně na sobě nahuštěny, což má za následek špatný kontrast obrazu. Na stejném obrázku vpravo je pak vidět *histogram* po *ekvalizaci*, kde už jsou nejčetnější jasové úrovně

roztaženy do jiných úrovní, což znamená že jednotlivé jasové úrovně jsou zastoupeny zhruba stejně četně a výsledný obraz má tak mnoho lepší kontrast.



Obr. 2 Ukázka ekvalizace histogramu a) Histogram původního obrazu b) Histogram ekvalizovaného obrazu

Ekvalizace tedy zvýší kontrast pro úrovně jasu blízko maxim histogramu a sníží kontrast blízko minim histogramu. V ideálním případě by měl *histogram* četnost jasových úrovní naprosto stejnou, ale v praxi tato vlastnost z důvodu malého počtu jasových úrovní a velkého počtu pixelů není možná. [4, s. 37]

2.3. Lokální filtrace a vyhlazování

Lokální předzpracování se podle účelu dělí na dva typy. Prvním typem je *vyhlazování*, které bude probráno v této kapitole. Hlavním úkolem *vyhlazování* je potlačit šum v obraze. Je tedy myšlenkově stejné, jako použití dolnofrekvenční propusti při zpracování ve frekvenční oblasti. Druhým typem je *detekce hran*. Tento typ má už podle názvu za úkol co nejvíce zviditelnit hrany, což připomíná hornofrekvenční filtr. *Detekce hran* bude popsána v kapitole 3.1.

Lokální filtrace šumu v obraze je založena na principu procházení obrazu bod po bodu a následné aplikace některé zvolené metody. Pokud je sejmutých více stejných obrazů, u kterých se mění pouze šum, tak je možnost jednoduchým průměrováním mezi body v jednotlivých obrazech, které se nacházejí na stejné pozici, docílit toho, že se tento šum z menší části odstraní. Hlavní výhodou tohoto postupu je minimální rozmazávání obrazu. Pokud ovšem je k dispozici pouze jeden obraz, který je potřeba filtrovat, tak je nutné k filtraci každého bodu použít jeho okolní body s využitím tzv. *konvoluce* a *konvoluční masky*. Hodnota počítaného bodu je pak nahrazena nově vypočítanou hodnotou, nebo druhou možností může být umístění této nově vypočítané hodnoty do nového obrazu. [4, s. 51-52]

2.3.1. Konvoluce a konvoluční maska

Konvoluční maska je matice obdélníkového, nebo čtvercového tvaru, která je vyplněna číselnými hodnoty podle typu metody. *Konvoluce* je operace podobná *korelaci*, jediný rozdíl je v tom, že u *konvoluce* je maska otočená o 180°. U *konvoluce* se tedy pomocí masky (neboli také jádra) hledá podobnost mezi sledovanou funkcí *f* a maskou *h*. Princip je takový, že se *maska* posouvá po jednotlivých řádcích a při každém posunutí se provede součin každého koeficientu *masky* s koeficientem jasu nacházejícím se „pod“ *maskou*, následně se tyto součiny sečtou a součet se pak může ještě dělit, nebo se rovnou zapíše do nového obrazu *g*. Ve většině případů se používají liché

velikosti *masky* z důvodu znalosti přesného středu. Rovnice (5) znázorňuje výpočet *konvoluční masky*, kde s_{max} a t_{max} značí polovinu *konvoluční masky* zaokrouhlenou směrem dolů (konkrétně s_{max} počet řádků a t_{max} počet sloupců), *x* zpracovávaný řádek obrazu, *y* zpracovávaný sloupec obrazu, *s* pozice řádku matice a *t* pozice sloupce matice.

$$g(x,y) = h * f = \sum_{s=-s_{\max}}^{s_{\max}} \sum_{t=-t_{\max}}^{t_{\max}} h(s,t) \cdot f(x-s, y-t) \tag{5}$$

Při *konvoluci* se na krajích obrazu stane, že nejsou k dispozici potřebná data. Například při *masce* velikosti 3x3 budou při zpracovávání bodu na levém okraji obrazu chybět hodnoty jasu pod hodnotami levého sloupce *masky*. Tento problém se dá řešit třemi způsoby. První způsob je zvětšení obrazu nulami o polovinu velikosti masky (tzv. „zero padding“). U druhého způsobu se celá *maska* pohybuje pouze v oblasti, kde má pod sebou potřebné hodnoty jasu. Výsledkem bude zmenšený obraz. Třetí způsob funguje tak, že ve zmiňovaných „kritických místech“ se aplikuje zmenšená *maska*, takže se pracuje pouze s koeficienty *masky*, které mají pod sebou jasovou hodnotu. [1, s. 57-60]

2.3.2. Lineární metody vyhlazování

Lineární metody využívají pro vyhlazování zvláštní třídu *lineárních filtrů*, kterým se říká *prostorově invariantní filtry*. Toto pojmenování vychází z jejich stálého chování při změně polohy filtru v obraze. Ovšem ve skutečnosti při zpracování obrazu je předpoklad linearity narušen kvůli tomu, že jsou hodnoty jasu nezáporné a omezené a také kvůli ohraničení obrazu, který tak není nekonečný. Z těchto dvou důvodů neplatí prostorová invariantnost pro všechny *konvoluční masky*. V praxi ale neexistuje jednoduchá teorie, která by věrohodně popisovala zmiňovaný problém, a tak se aplikuje klasický lineární přístup a následně se zajišťuje, aby na okrajích obrazu nevznikaly problémy, některou vhodně zvolenou metodou, která byla probrána na konci kapitoly 2.3.1. [4, s. 53]

Nejjednodušší *lineární metodou* je tzv. *vyhlazování průměrováním*. Výsledkem každé *konvoluce* je průměr hodnot jasů ze sousedních bodů. [1, s. 62] Rovnice (6) znázorňuje podobu *konvoluční masky* velikosti 3x3. [4, s. 53]

$$h = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{6}$$

Další metoda vycházející z *průměrování* se nazývá *Gaussovo vyhlazování*. Toto vyhlazování je inspirování *Gaussovou křivkou* což znamená, že hodnoty blíže středu masky mají vyšší váhu oproti ostatním. [1, s. 62] Příklad takové masky zobrazuje rovnice (7).

$$h = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{7}$$

Metod *průměrování* se u vyhlazování příliš nepoužívá z důvodu rozmazávání obrazu, a tak využití nacházejí ve větší míře například u výpočtu střední hodnoty jasu. [4, s. 54]

U *konvolučních matic* s většími rozměry může docházet k velké časové náročnosti výpočtu vyhlazování. Tento čas výpočtu se může výrazně zkrátit díky vlastnosti zvané *separabilita*, která *konvoluční matici* rozdělí na součin jedné řádkové a jedné sloupcové matice a vyhlazování se pak provádí jako součin nejdříve s jednou maticí a pak s maticí druhou. [1, s. 64]

$$\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 4 \\ 6 \\ 4 \\ 1 \end{bmatrix} \tag{8}$$

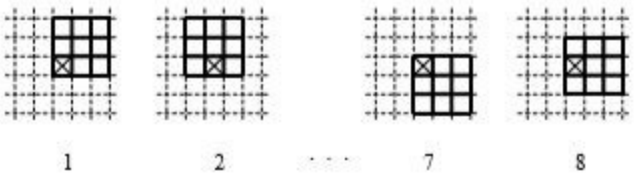
Rozložení a vzhled takového *separabilního filtru* je možné vidět v rovnici (8). U výpočtu konvoluce při použití standartní konvoluční matice by bylo zapotřebí 25 násobení a následně 24 sčítání u každého pixelu, zatímco při použití *separabilního filtru* se tyto čísla redukují na 10 součinů a 8 součtů, což ve výsledku znamená velké zrychlení filtrace.

Zajímavou skupinu prostorově invariantních filtrů tvoří tzv. *rekurzivní filtry*. Jejich hlavní zajímavostí je to, že při konvoluci počítají s vypočtenou hodnotou u předchozí polohy masky doplněnou o původní hodnoty v nové poloze, zatímco *nerekurzivními filtry* pracují vždy pouze s původními nezměněnými hodnotami. *Rekurzivní filtry* jsou mnohem efektivnější než nerekurzivní, ale řešení těchto filtrů je velice složité, jelikož při návrhu je potřeba pracovat s pamětí, která si bude pamatovat minulé hodnoty, což znamená že se jedná o dynamický systém. Z důvodu kombinování současných a minulých hodnot je ve filtru zavedená zpětná vazba z čehož vyplývá, že při návrhu je potřeba řešit jeho stabilitu. Dále je potřeba nalézt správný směr zpracování, jelikož neexistuje teorie, který směr by měl být preferován, tak jako je čas u 1D signálů. Také se musí zabránit, aby po filtraci vznikal fázový posun, který by mohl zapříčinit posouvání hran a jiných důležitých bodů v obraze atd. U 1D signálu je jasná teorie, jak vyrobit *rekurzivní filtr*, ale u 2D obrazu tomu tak bohužel není, takže se volí spíše cesta kaskádního řazení jednoduchých filtrů. [4, s. 54-55]

2.3.3. Nelineární metody vyhlazování

Nelineární metody částečně eliminují hlavní nevýhodu *lineárních filtrů*, kterou je rozmazávání hran. Jejich princip závisí na tom, že se snaží v okolí každého bodu nalézt pouze tu část, která má podobnou hodnotu jasu jako zpracovávaný bod a jen tyto pixely budou využity pro zpracování daného bodu.

Klasickým případem *nelineární filtrace* je *metoda rotující masky*. U této metody se počítá rozptyl jasů v osmi polohách masky rotující okolo počítaného bodu. Potom se jednotlivé rozptyly mezi sebou porovnávají a vybere se maska s nejmenším rozptylem, která se použije pro výpočet aritmetického průměru a následného zapsaní hodnoty jasu do nového obrazu.



Obr. 3 Rotace masky

Na *Obr. 3* je znázorněna rotace masky velikosti 3x3 okolo počítaného bodu. Velikost a tvar výrazně ovlivňuje rychlost výpočtu filtrace v obraze. Tato metoda nerozmazává hrany, ale právě naopak má mírně ostřící charakter.

Dalším typem nelineární metody je tzv. *průměrování s omezením změn jasu*. Tato metoda funguje jako klasická lineární metoda, což znamená že má svou masku, která se posouvá bod po bodu v obraze a následně vypočítá průměr hodnot. Její rozdíl je ale v tom, že je u ní povolena pouze omezená změna hodnoty oproti hodnotě původní. To znamená, že pokud je diference mezi nově vypočítanou hodnotou a hodnotou původní v předem určené prahové mezi, tak se do nového obrazu zapíše nově vypočítaná hodnota. Pokud ale je diference větší než povolená mez, pak je do nového obrazu zapsána hodnota původní. Díky tomu nedochází k rozmazání hran v původním obraze, a tak je tato metoda velice účinná pro opravu velkoplošných chyb. [4, s. 55-56]

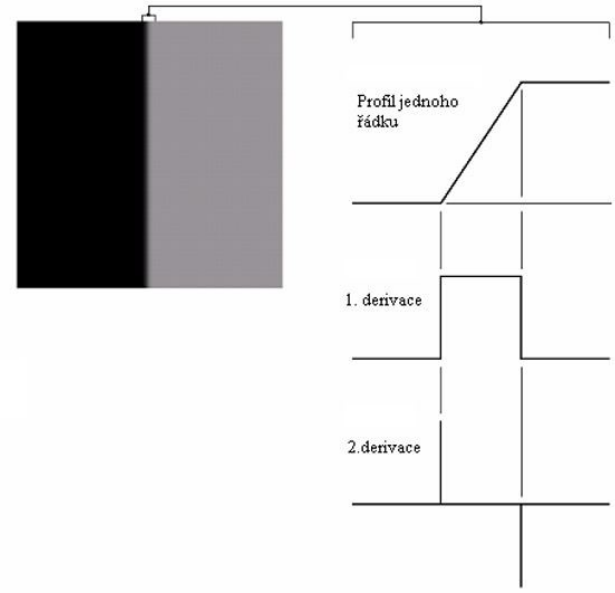
Posledním zde zmíněným nelineárním filtrem bude *statistický filtr,* u kterého je využito metody *mediánové filtrace*. *Statistický filtr* má velice dobré vlastnosti při potlačování šumu, kterému se říká „pepř a sůl“. Tento šum má v šedém obraze podobu jedno pixelových jasných černých a bílých teček, které už na pohled vypadají, že jsou v obraze nežádané. Princip metody *mediánové filtrace* vychází z teorie výpočtu klasického *mediánu* využívaného v matematické *statistice*. Prakticky *statistický filtr* funguje tak, že se maska o předem zvolených rozměrech obdobně jako u předchozích případů pohybuje po obraze, ale na rozdíl od ostatních u této metody neprobíhá konvoluce. Místo toho se z hodnot nacházejících se pod maskou vypočítá *medián* a jeho hodnota je poté uložena místo počítaného bodu v novém obraze. V některých aplikacích se také používá místo *mediánu* některý, podle velikosti matice vhodně zvolený, *k*-tý prvek v posloupnosti. [1, s. 64] U *mediánové filtrace* je důležité vhodně zvolit tvar a velikost masky, nebo v některých aplikacích tuto metodu vůbec nepoužívat, jelikož by se mohlo stát, že by filtr mazal tenké hrany, čáry, nebo ostré rohy. [4, s. 57]

3. Rozpoznání objektů v obraze

Rozpoznání neboli také *identifikace* různých objektů bývá hlavní náplní při zpracování obrazu. Po tom, co je z obrazu vyfiltrován šum pomocí některé z metod zmíněných v kapitole 2, nastupuje úloha *segmentace* a *rozpoznání hran* pomocí různých algoritmů, které budou probrány v této kapitole. [2, s. 213] V různých aplikacích je možné rozpoznávat prakticky téměř cokoliv. Ať už se jedná o různé tvary, velikosti, umístění v obraze, podobnost dvou objektů a mnoho dalších. Objekty se dají identifikovat buď pomocí *detekce hran* anebo pomocí *segmentování* jednotlivých objektů reprezentovaných v obraze. Jelikož každé těleso by mělo být obklopeno svou hranicí, tak by se dalo říci, že identifikace jak pomocí *detekce hran*, tak pomocí *segmentování*, by měla mít stejné výsledky. V praxi tomu ovšem tak vždy není, jelikož některé objekty nemusí být zcela zřetelné, nebo se může projevovat velký šum, a tak potom poskládané identifikované hrany nemusí dávat uzavřenou smyčku objektu. [3, s. 84]

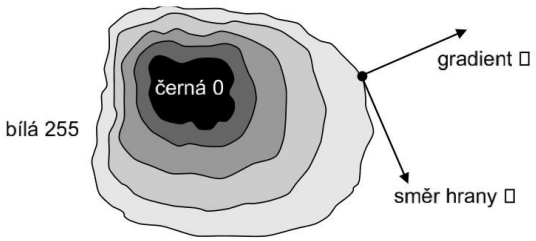
3.1. Detekce hran

Výsledkem *detekování hran* by měl být obraz ve kterém budou pouze *detekované hrany*, a tak z obrazu zmizí nezajímavé pozadí. [2, s. 213] *Hrana* je teoreticky hranice rozpoznávaného objektu a měla by se projevovat jako prudká změna jasové hodnoty mezi sousedícími pixely v obraze. Ovšem v praxi nejsou pořízené obrazy úplně dokonale vyfiltrované, a tak *hranám* spíše odpovídá přechodná změna jasových hodnot (*Obr. 4*). [5, s. 45-46]



Obr. 4 Ukázka profilu jednoho řádku skutečné hrany, 1. a 2. derivace této hrany

První technikou pro *detekci hran* je využití první derivace, kde změnu funkce udává jejich gradient. [4, s. 57] Z průběhu první derivace na *Obr. 4* je vidět, že první derivace je kladná v místě hrany a nulová v místech konstantního jasu. [5, s. 46] Často je potřeba vědět pouze velikost gradientu, protože pixely, které mají největší model gradientu ve svém okolí, tvoří hrany. Ovšem pokud by byla potřeba zjistit i směr hrany, tak tento směr je vždy kolmý na směr vektoru gradientu což znázorňuje *Obr. 5.* [4, s. 58]



Obr. 5 Vektor gradientu kolmý vzhledem k směru hrany

K *detekci hran* se dá využít také druhé derivace, která představuje rychlost změny hodnot jasu. [1, s. 68] Z průběhu druhé derivace na *Obr. 4*. je vidět, že na tmavé části *hrany* je kladný impuls a na světlé části zase záporný impuls, čehož se dá k detekci využít. Pro detekci *hran* s využitím druhé derivace je důležité, aby nebyl obraz příliš zašumělí, jinak by mohlo docházet k snadné ztrátě těchto dvou impulzů mezi šumem. [5, s. 46-47] Gradientních operátorů se kromě *detekce hran* dá také využít pro ostření obrazu, což ve frekvenčním zpracování obrazu odpovídá zdůraznění vysokých frekvencí. [4, s. 58]

3.1.1. Konvoluční masky vycházející z derivací obrazové funkce

V této kapitole budou uvedeny *operátory*, které lze vyjádřit pomocí *konvoluční masky*, a tak pro jejich výpočet je možné využít klasickou *konvoluci*.

Nejstarším a zároveň také nejjednodušším operátorem je *Robertsův operátor*. Jeho masky, které lze vidět v rovnici (9), mají rozměr pouze 2x2, a tak tento operátor není příliš výhodný, jelikož se projevuje jeho citlivost na šum.

$$h_1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad h_2 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{9}$$

Dalším operátorem je *Sobelův operátor*, který už má velikost masky 3x3 a jeho konvoluční masky zobrazuje rovnice (10). Tento operátor se využívá hlavně pro detekci svislých a vodorovných hran pomocí masek h_1 a h_3.

$$h_1 = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \qquad h_2 = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix} \qquad h_3 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{10}$$

Operátor Prewittové stejně jako *Sobelův*, *Kirschův* a *Robinsův* aproximuje první derivaci obrazové funkce. [4, s. 60-61] Jejich masky jsou tvořeny pro osm možných směrů a vybrána je potom ta, která má největší modul gradientu. *Operátor Prewittové* odstraňuje nevýhodu *Sobelova* operátoru, kterou je absence vyhlazovacích modulů, takže *operátor Prewittové* obsahuje vektorovou hodnotu poskytující vyhlazování. [2, s. 214] *Operátor Prewittové* zobrazuje rovnice (11), *Kirschův* rovnice (12) a *Robinsův* rovnice (13). Všechny tyto operátory mají masky pro 8 směrů, ale pro ušetření místa zde budou uvedeny pouze první tři, ostatní se dají jednoduše odvodit rotací prvků v matici.

$$h_1 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \qquad h_2 = \begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix} \qquad h_3 = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \tag{11}$$

$$h_1 = \begin{bmatrix} 3 & 3 & 3 \\ 3 & 0 & 3 \\ -5 & -5 & -5 \end{bmatrix} \qquad h_2 = \begin{bmatrix} 3 & 3 & 3 \\ -5 & 0 & 3 \\ -5 & -5 & 3 \end{bmatrix} \qquad h_3 = \begin{bmatrix} -5 & 3 & 3 \\ -5 & 0 & 3 \\ -5 & 3 & 3 \end{bmatrix} \tag{12}$$

$$h_1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -2 & 1 \\ -1 & -1 & -1 \end{bmatrix} \qquad h_2 = \begin{bmatrix} 1 & 1 & 1 \\ -1 & -2 & 1 \\ -1 & -1 & 1 \end{bmatrix} \qquad h_3 = \begin{bmatrix} -1 & 1 & 1 \\ -1 & -2 & 1 \\ -1 & 1 & 1 \end{bmatrix} \tag{13}$$

Posledním zde zmíněným operátorem bude *Laplaceův operátor*, který jako jediný pracuje s druhou derivací obrazové funkce což znamená, že je invariantní vůči otočení a udává pouze velikost hrany, nikoliv její směr. Dvě hlavní konvoluční jádra pro okolí 3x3 zobrazuje rovnice (14), kde konvoluční maska h_1 je pro čtyř sousedství a maska h_2 pro osmi sousedství. [4, s. 60] Konvoluční maska se může také využít v negativní variantě, kdy pozadí zůstane černé a hrany budou zobrazeny bílými čarami. Toho se docílí pouhým otočením znamének u masek v rovnici (14). [1, s. 69]

$$h_1 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad h_2 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{14}$$

V některých případech může být vhodné využití *Laplaciánu s větší vahou pixelů blíže reprezentativnímu bodu masky* (rovnice (15)). Nevýhodou *Laplaceova operátoru* bývá velká citlivost na šum a také dvojitá odezva na hrany, které představují tenké linie v obraze. [4, s. 60]

$$h_1 = \begin{bmatrix} 2 & -1 & 2 \\ -1 & -4 & -1 \\ 2 & -1 & 2 \end{bmatrix} \qquad h_2 = \begin{bmatrix} -1 & 2 & -1 \\ 2 & -4 & 2 \\ -1 & 2 & -1 \end{bmatrix} \tag{15}$$

Laplaceova operátoru se dá také využít pro ostření obrazu a zvýraznění hran. Toho se docílí pomocí operace zvané superpozice, při které je jednoduše přičtena hodnota derivace přímo k originálnímu obrazu. Konvoluční masku tohoto ostření obrazu zobrazuje rovnice (16). [1, s. 69]

$$h_1 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{16}$$

3.1.2. Hrany jako průchody nulou druhé derivace

Vzhledem k nevýhodám klasického samotného *Laplaceova operátoru* byl vytvořen tzv. *Laplacian of Gaussian operátor*, který pro detekci hran využívá průchodu nulou druhé derivace, přičemž kombinuje vlastnosti *Gaussova* a *Laplaceova* operátoru. [1, s. 69] Prakticky tato metoda funguje tak, že je obrazová funkce rozmazána *Gaussovým vyhlazováním* a poté je pro odhad druhé derivace použit *Laplaceův operátor*. Konvoluční masku pro *operátor Laplacian of Gaussian* zobrazuje rovnice (17). Tento operátor má ovšem i své nevýhody, jaké jsou například rozmazávání příliš ostrých hran, nebo spojování hran do uzavřených křivek, což bylo dříve označováno spíše za výhodu, ale v mnoha aplikacích tato vlastnost působila značné potíže.

$$h_1 = \begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ -2 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix} \tag{17}$$

3.2. Segmentace obrazu – Prahování

Segmentací obrazu se rozumí operace, pomocí kterých se dá zjistit, zda jsou v obraze objekty, které jsou hledány například pro analýzu, zpracování, nebo rozpoznání, podle příslušné aplikace. Další úlohou *segmentace* může být také odstranění nepotřebných objektů a částí obrazu, které už v dalším zpracování nejsou potřebné. Vzhledem k faktům, že se některé objekty mohou překrývat, pozadí může mít příliš hran, které se promítnou k sledovaným objektům atd., je úloha *segmentace* obrazu tou nejtěžší úlohou ve zpracování obrazu.

Prahování neboli také *thresholding* se používá pro převedení obrazu s více stupni odstínu šedé do obrazu. ve kterém se vyskytují pouze černé a bílé pixely. Hlavní úlohou *prahování* je nalezení optimálního *prahu* tak, aby po této operaci zůstaly viditelné pouze potřebné hrany a ostatní zmizeli. V dalším kroku jsou pak převedeny hodnoty větší než hodnota *prahu* na bílou, a naopak menší hodnoty na černou, nebo i obráceně. Určení *prahu* je možné několika způsoby. Tím nejjednodušším je vybrat práh pomocí histogramu, nebo také vyzkoušením několika možných hodnot, dokud není nalezena hodnota s nejlepším výsledkem. Dalším způsobem je nalezení *prahu* automaticky podle některé z metod probraných v následujících podkapitolách. [1, s. 77-78]

3.2.1. Metoda automatického nalezení prahu

Metoda *automatického nalezení prahu* se považuje za tu vůbec nejjednodušší. Hlavním předpokladem je vhodné rozdělení jasových hodnot což znamená, že by histogram obrazu představoval dva vrcholy, kde by jasové hodnoty jednoho vrcholu odpovídaly pozadí a jasové hodnoty druhého vrcholu zase popředí, tedy hledaným objektům. Postup této metody je následující. Nejdříve je zvolena počáteční hodnota prahu T, která se obvykle volí jako střed mezi minimální a maximální hodnotou jasu v obraze. Ve druhém kroku je rozdělen obraz na dvě množiny. První množina A odpovídá popředí s hodnotami jasu většími nebo rovny hodnotě T a druhá množina B zase odpovídá pozadí s hodnotami jasu menšími než T. V dalším kroku se pak vypočítá průměrná hodnota jasu z množiny A (m_A) podle rovnice (18) a průměrná hodnota jasu z množiny B (m_B) podle rovnice (19), kde k značí jasovou úroveň, L počet jasových úrovní a normalizovaný histogram $p_k = n_k / N$, kde n_k je počet bodů o jasové úrovni k a N je celkový počet bodů v obraze.

$$m_A = \sum_{k=T}^{L-1} kp_k \tag{18}$$

$$m_B = \sum_{k=0}^{T-1} kp_k \tag{19}$$

Po určení průměrů se vypočítá nová hodnota prahu jako $T_n = (m_A + m_B)/2$. Pokud je rozdíl mezi zvoleným a vypočítaným prahem $|T - T_n| > 0{,}5$, tak se T_n nastaví jako T a počítá se znovu nová hodnota T_n od druhého kroku této metody. [1, s. 78-79]

3.2.2. Nalezení optimálního prahu – Otsuova metoda

Otsuova metoda je další metodou automatického nalezení prahu. Tato metoda má použití hlavně v aplikacích, kde není podle histogramu jasně definován vrchol pro pozadí jasových hodnot a vrchol pro popředí jasových hodnot což znamená, že pomyslné dva vrcholy spolu částečně splývají, a tak není možné jasně určit prahovou hodnotu *T.* Teoretické řešení *Otsuovy metody* je velice složité, jelikož se na histogram nahlíží jako na dvě spojité křivky, kde jedna křivka tvoří pozadí a druhá popředí. Podstata teoretického řešení spočívá ve výpočtu chybných klasifikací pozadí jako popředí a obráceně, kde se na základě spojitého histogramu počítají pravděpodobnosti výskytu popředí a pozadí, což je z hlediska výpočtu příliš náročné a nepraktické.

Ovšem na druhou stranu *Otsu* ukázal i praktické řešení, které vychází z faktu, že histogram je ve skutečnosti diskrétní funkce, a tak je výpočetní postup o mnoho jednodušší. U *Otsuové metody* je počítána hodnota prahu *T* na základě histogramu $p_k = n_k / N$, kde n_k je počet bodů o jasové úrovni k a N je celkový počet bodů v obraze. Vzhledem k hodnotě prahu *T* je histogram rozdělen na množinu C_0 představující pozadí a množinu C_1 představující pozadí. Množinu C_0 pak představují sumy ω_0, μ_0 a množinu C_1 sumy ω_0, μ_0 počítané podle následujících rovnic a závislé na prahové hodnotě *T*.

$$\omega_0 = \sum_{k=0}^{T-1} p(k) \tag{20}$$

$$\mu_0 = \sum_{k=0}^{T-1} kp(k) / \omega_0 \tag{21}$$

$$\omega_1 = \sum_{k=T}^{L-1} p(k) \tag{22}$$

$$\mu_1 = \sum_{k=T}^{L-1} kp(k) / \omega_0 \tag{23}$$

Následně se ještě vypočítá střední hodnota v původním rozdělení podle rovnice (24) a nakonec se nalezne optimální práh jako takové *T*, které maximalizuje výraz σ (mezitřídní rozptyl) v rovnici (25). [1, s. 79-81]

$$\mu_T = \sum_{k=0}^{L-1} kp(k) \tag{24}$$

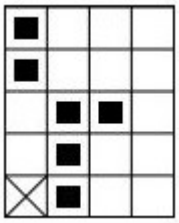
$$\sigma^2 = \omega_0(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_T)^2 \tag{25}$$

3.3. Matematická morfologie

Pojem *Matematická morfologie* vychází z matematických nelineárních operací, které jsou využívány pro segmentaci, a hlavně pro předzpracování obrazu. Jejich operátory se využívají v případech, kde je požadavek na rychlé zpracování. *Matematická morfologie* se používá převážně

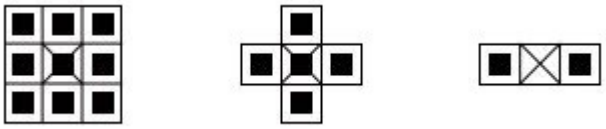
u obrazů s binárními hodnotami jasu (černá a bílá) a nevyužívá klasické konvoluce jako metody uvedené v předchozím textu, ale používá vlastnosti bodových množin a jejich podmnožin.

Binární obraz lze vyjádřit jako 2D bodovou množinu. Body objektů v obraze reprezentují množinu X, což odpovídá pixelům s hodnotou jedna (černé čtverce v *Obr. 6*). *Body doplňku* X^c *popisují pozadí a reprezentují se pixely s hodnotou nula* (bílé čtverce v *Obr. 6*). *Počátek* (označen na *Obr. 6* křížkem) *má souřadnice (0,0) a souřadnice ostatních bodů (x, y) mají stejný význam, jako je v matematice obvyklé.*



Obr. 6 Příklad bodové množiny

Morfologická transformace Ψ je dána relací mezi obrazem (bodová množina X) s jinou, typicky menší bodovou množinou B, které se říká strukturní element. Strukturní element B je vztažen k "lokálnímu" počátku O, kterému se říká reprezentativní bod. Některé typické strukturní elementy jsou na Obr. 7.

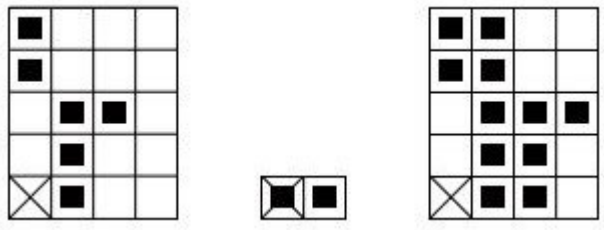


Obr. 7 Příklady strukturních elementů

Strukturní element se pak posouvá po jednotlivých bodech od počátku v obraze a výsledek relace mezi elementem a obrazem se podle jejich matematického předpisu zapíše do výstupního obrazu. Některé důležité operace *matematické morfologie* mezi bodovou množinou vstupního obrazu a strukturním elementem jsou uvedeny v následujících podkapitolách. [4, s. 71-72]

3.3.1. Dilatace

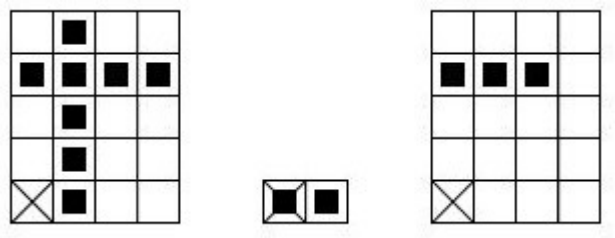
Výsledkem *Dilatace* je vždy bodová množina, která odpovídá vektorovému součtu všech možných bodů množiny *X* se strukturním elementem *B*. Pro jasnou představu je na *Obr. 8* ukázána vstupní bodová množina (vlevo) po *dilataci* se strukturním elementem (uprostřed), kde výsledek této operace je zobrazen vpravo. *Dilatace* se využívá hlavně pro zaplnění malých děr, úzkých zálivů, nebo ke zvětšení objektů. [4, s. 73-74]



Obr. 8 Dilatace

3.3.2. Eroze

Eroze je duální operace k *dilataci* a používá se pro zjednodušení a ztenčení struktury objektů. Princip *eroze* je v tom, že pro každý bod obrazu se ověřuje, zda v daném bodě a v jeho okolí se nacházejí body objektu tak jak předepisuje strukturní element. Pokud je podmínka splněna, zapíše se v erodovaném bodě do výsledného obrázku 1 a v opačném případě 0. Příklad erodované bodové množiny ukazuje *Obr. 9*.



Obr. 9 Eroze

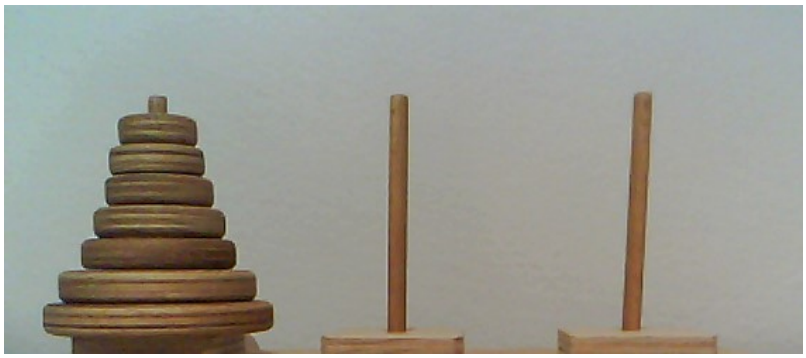
3.3.3. Hledání kostry obrazu

Kostrou obrazu se rozumí linie široká jeden pixel, která se nachází vždy ve středu mezi hranicemi objektu, nebo čáry tvořící objekt. Pro hledání *kostry* se obvykle používají tzv. *ztenčovací algoritmy*. Po použití těchto algoritmů bývá požadováno, aby počet objektů a děr v obrazu zůstal stejný jako v původním, aby poloha koncových bodů zůstala zachována anebo aby odstranění bodu nezpůsobilo přerušení čáry, což bývá v některých případech obtížné. K hledání *kostry* lze přistupovat dvěma způsoby. Tím prvním je způsob pomocí *erozí* a *dilatací*, který ovšem často neztenčuje na tloušťku jeden pixel a nezachovává *homotopii*. Druhým způsobem je tzv. *homotopické ztenčování*, které bývá velmi často tou lepší volbou vzhledem k tomu, že splňuje dříve zmiňované podmínky pro hledání *kostry* a také zachovává *homotopii* objektů.

Při *homotopickém ztenčování* objektů tvoří vstup dvoubitový obraz. Algoritmus prochází vstupní obraz a přitom zkoumá, zda jednotlivé body objektu budou označeny za kandidáty na odstranění či nikoliv. K finálnímu ztenčení na jeden pixel je potřeba více průchodů obrazem, takže se obraz prochází neustále dokola dokud už není možné odstranit žádný pixel. Aby byla *kostra* správně centrována jsou tyto průchody rozděleny na liché a sudé. Jednotlivé podmínky a postupy pro označení bodu jako kandidáta se liší, ale ve svém jádře vychází z postupu, že se nejdříve u zkoumaného bodu objektu zjišťuje počet bodů objektů v jeho okolí. Dále se zjistí přechodové číslo, které odpovídá přechodům z 0 do 1 a obráceně v jeho okolí. A nakonec pokud jsou předchozí dvě podmínky splněny a okolní body odpovídají některé z předdefinovaných logických rovnic podle sudého nebo lichého průběhu, se zkoumaný bod přepíše na nulu. Pokud některá z podmínek není platná tak se zkoumaný bod ponechá jako 1.

4. Hanojské věže

Pro praktickou část bakalářské práce byla zadána realizace „kamerového rozhodčího“ ke hře *Hanojské věže*, která by měla být naprogramována v programovacím jazyce C#. *Hanojské věže* jsou matematicko-logickou hrou, u které se nachází tři tyče a 7 různě širokých kotoučů. V základní poloze jsou tyto kotouče seřazeny od nejmenšího po největší na levé tyči (*Obr.10*) a cílem hry je takto seřazenou věž přeskládat ve stejném pořadí buď na prostřední tyč, nebo na tyč pravou, přičemž při řešení je možné vždy přesunou pouze jeden kotouč a v žádném případě se nesmí nacházet širší kotouč výše než užší. Cílem rozhodčího tedy bylo sledování a lokalizace jednotlivých kotoučů, přičemž by bylo vyhodnoceno, jestli nedošlo k chybě při řešení úlohy.

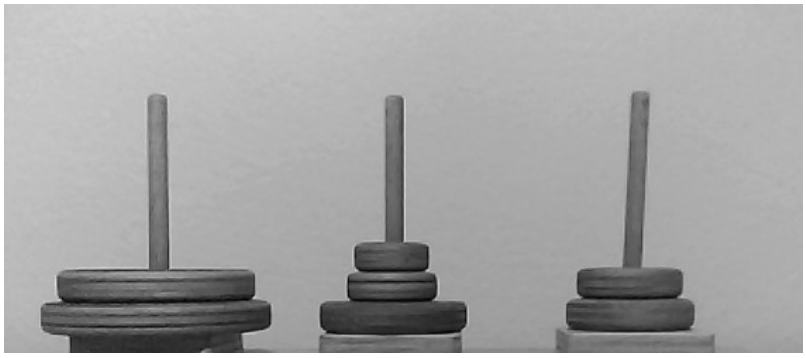


Obr. 10 Hanojské věže

4.1. Zpracování obrazu

Prvním důležitým bodem bylo vymyslet vhodný postup zpracování snímané scény, který by vedl k co nejdokonalejšímu černobílému obrazu, se kterým by byla co nejspolehlivější a nejjednodušší práce při identifikaci jednotlivých objektů. Z velkého množství filtrů, hranových detektorů apod., které jsou uvedeny v předchozích kapitolách, nebylo možné jednoznačně vybrat, který z typů a postupů by byl nejvhodnější. Z tohoto důvodu byla nejdříve naprogramována aplikace *Testování filtrů*, která se nachází v příloze na CD. V této aplikaci se po připojení kamery zobrazí v okně živý obraz snímaný kamerou. Po zmáčknutí tlačítka je poté jeden tento snímek přesunut do nového okna, ve kterém je možné na něj aplikovat kteroukoliv naprogramovanou metodu v libovolném pořadí. Takto zpracované obrazy je poté možné přesunout do dalších předpřipravených oken, takže je možné jednotlivé postupy zpracování obrazu mezi sebou porovnávat a vybrat ten nejlepší. Pro porovnávání bylo také možné využít naprogramované zobrazení histogramu obrazu.
V následujících podkapitolách bude uvedeno, jak jednotlivé metody působí na snímaný obraz, který je uveden na *Obr.11*. Prvním krokem u zpracování obrazu je vždy převedení barevného obrazu na černobílý. Tato operace se provádí z důvodu jednodušších a výpočetně méně náročných operací, protože u barevného obrazu tvoří každý pixel tři barevné hodnoty (červená, zelená, modrá), zatímco u černobílého má každý pixel pouze jednu hodnotu jasu. Rovnic pro převod barevného obrazu na černobílý je velké množství, protože vždy záleží, kterou barvu je potřeba z obrazu zvýraznit, a tak je jí přiřazena větší jasová hodnota. Pro zdejší zpracování byla vybrána rovnice (26), kde F je výsledná úroveň jasu, R je červená složka barvy, G zelená a B modrá.

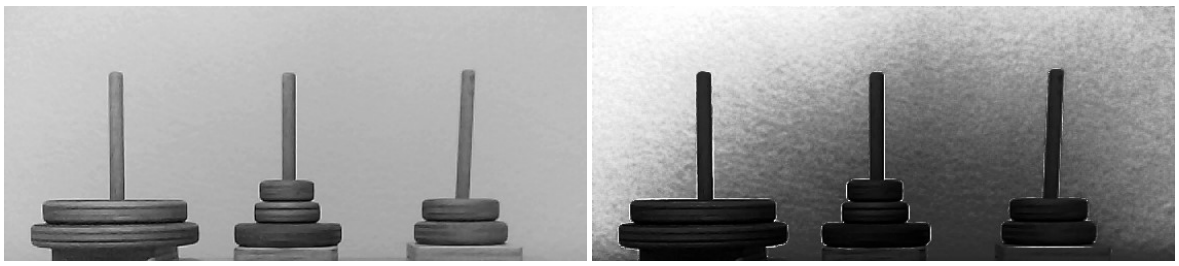
$$F = 0{,}21 \cdot R + 0{,}72 \cdot G + 0{,}07 \cdot B \qquad (26)$$



Obr. 11 Původní černobílý obraz

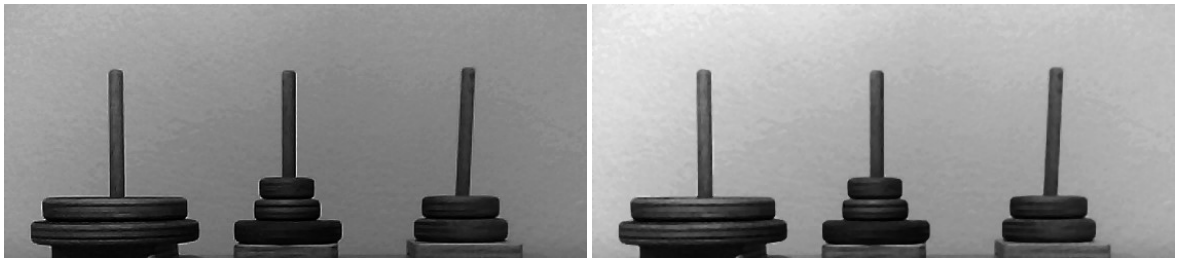
4.1.1. Zlepšení kontrastu

Hlavní myšlenkou zlepšení kontrastu v obraze bývá, že po této operaci se hledané objekty zvýrazní a nezajímavé pozadí lehce zmizí. Za účelem zlepšení kontrastu při zpracování obrazu byly naprogramovány tři metody, konkrétně *roztažení kontrastu* (*Obr. 12 a*), *vyrovnání histogramu* (*Obr. 12 b*) a *mocnění jasových hodnot* (*Obr. 13 a*). Z obrázku *roztažení kontrastu* lze vidět, že tato metoda není příliš účinná, protože nedošlo k žádnému výraznému zlepšení. Důvod je takový, že tahle metoda nijak neupravuje jasovou četnost pixelů, ale pouze roztáhne rozmezí, ve kterém se tyto četnosti pohybují, na maximum. A jelikož jsou jasové hodnoty pixelů vstupního obrazu už tak poměrně roztáhlé, tak nemůže dojít k žádné změně. K této změně však už značně dochází v případě metody *vyrovnání histogramu*. Jak je uvedeno v kapitole 2.2.2., tak takhle metoda už pracuje i s četnostmi pixelů u jednotlivých jasových úrovní, takže výsledek oproti původnímu obrazu je znatelný. Bohužel se ale u téhle metody nepříznivě ukázalo nerovnoměrné osvětlení snímané scény, takže i pozadí obrazu má poměrně nejednotné jasové hodnoty, což by mohlo vadit v dalších krocích.



Obr. 12 a) Roztažení kontrastu b) Vyrovnání histogramu

Metoda *mocnění jasových hodnot* funguje na principu umocnění všech jasových hodnot na druhou a jejich následného převedení pomocí obyčejné trojčlenky zpět do rozmezí jednoto bytu. Výsledek této metody byl poměrně příznivý, jelikož hledané objekty poměrně značně ztmavly, ale na druhou stranu lehce ztmavlo i pozadí, což je nežádané. Nicméně při testování byl zjištěn zajímavý jev. Pokud se před aplikováním metody *mocnění jasových hodnot* aplikuje na obraz některý z vyhlazovacích filtrů, tak po zlepšení kontrastu zůstanou hledané objekty stejně tmavé jako v předchozím zmíněném případě (*Obr.13 a*), ale pozadí výrazně zesvětlá, což je pro další zpracování vhodné, takže se tato metoda po aplikaci vyhlazovacího filtru ukázala jako velmi dobrá. Na *Obr. 13 b* je pro filtraci zvoleno *vyhlazování mediánem*.

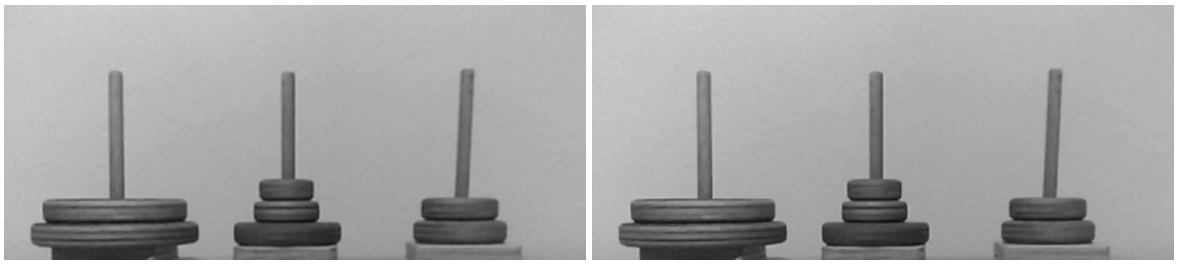


Obr. 13 a) Mocnění jasových hodnot b) Mocnění jasových hodnot po filtraci

4.1.2. Obrazové filtry

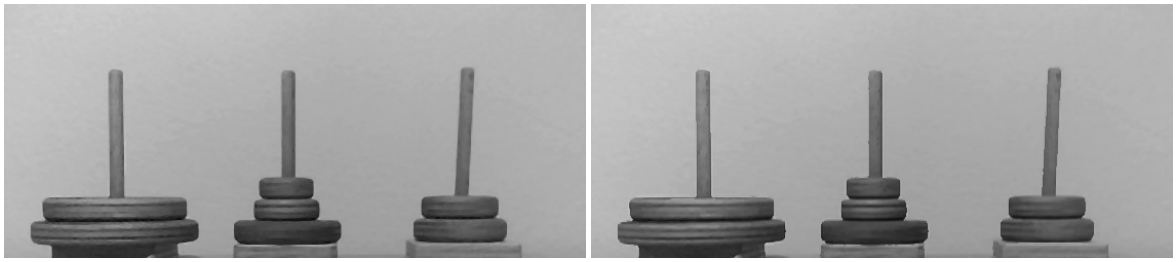
Jelikož pro realizaci bakalářské práce byla zvolena obyčejná webkamera, tak pořizovaný obraz nebyl příliš kvalitní, a tak jej bylo zapotřebí vyfiltrovat. Co se týče použitých filtrů, tak všechny zde zmíněné filtry rozmazávají původní obraz, nicméně u filtrů je hlavně důležité, aby vyfiltrovaly některý z typů šumů, který se v obraze nachází, a tak lehké rozmazání by nemělo vadit. V detailním pohledu na *Obr.11* si lze všimnout, že okolo hran se nachází podivné bílé pruhy, které hlavně vadí v hranové detekci, jelikož vytváří dvojité hrany, a tak je bylo nutné některým vhodně zvoleným filtrem odstranit.

Jelikož obraz *Hanojských věží* není příliš složitý, tak veškeré filtry spolehlivě odstranily výše zmíněný šum, a tak se lišily jen stupněm, jakým rozmazaly veškeré hrany, a tedy i zpracovávaný obraz. S největším rozmazáním se potýkaly lineární filtry, kde nejvíce rozmazaný obraz je u *metody průměrování* (*Obr.14 a*), kdy se stupeň rozmazání ještě zvětšuje s rozměry matice. Druhá lineární metoda, kterou je *Gaussovo vyhlazování* (*Obr.14 b*), je už vzhledem k rozmazání hran a obrazu poměrně šetrnější a jen trochu zaostává oproti nelineárním filtrům.

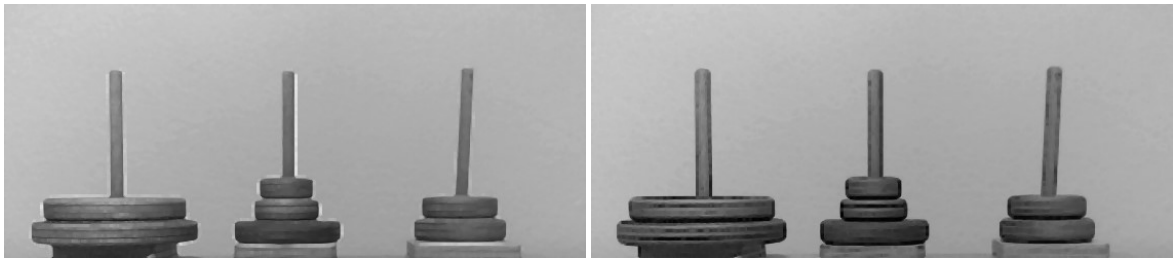


Obr. 14 Lineární filtry a) Metoda průměrování b) Gaussovo vyhlazování

I přes to ale *mediánový filtr* (*Obr.15 a*), který je nelineárním filtrem, je stále kvalitnější, jelikož méně rozmazává hrany. Co se týče metody *k-tého prvku* v posloupnosti, tak výsledky při výběru hodnot okolo *mediánu* se příliš od *mediánové filtrace* nelišily, ovšem při výběru hodnoty v okolí maximálního prvku (*Obr.16 a*) jsou už změny patrné a v okolí některých hran se objevují černé pixely. Naopak při výběru v okolí minimálního prvku (*Obr.16 b*) je vidět zase větší výskyt bílých pixelů. Posledním naprogramovaným nelineárním filtrem byl filtr využívající metody *rotující masky* (*Obr.15 b*), který se ukázal jako nejšetrnější k rozmazání hran, avšak na druhou stranu byl výstupní obraz lehce kostkovaný.



Obr. 15 Nelineární filtry a) Filtrace mediánem b) Metoda rotující masky

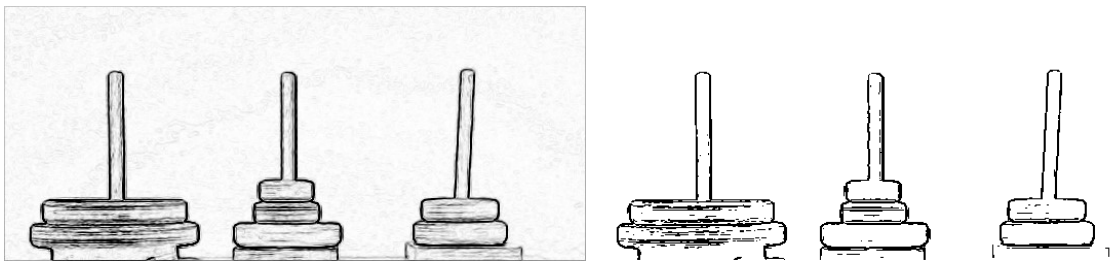


Obr. 16 Metoda k-tého prvku a) Volba druhé maximální hodnoty v matici b) Volba druhé minimální hodnoty v matici

4.1.3. Hranové detektory

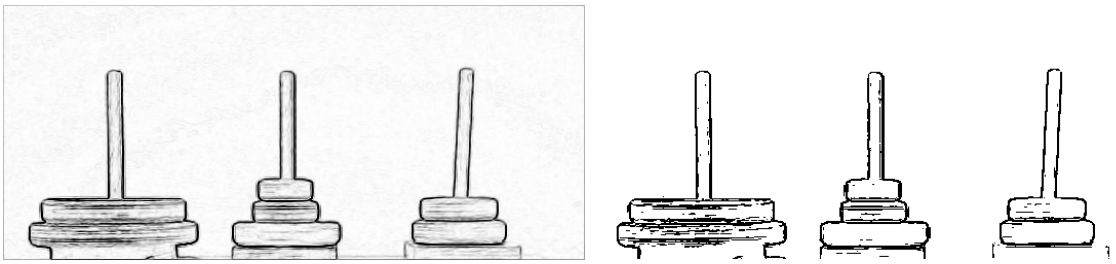
Pro pozdější práci s obrazem je velice důležité zvolit vhodný hranový detektor, jelikož na jeho základě se později bude odvíjet složitost použitých metod automatického nalezení prahu a také matematické morfologie pro doladění obrazu k dokonalosti. V této podkapitole budou uvedeny jednotlivé naprogramované hranové detektory, které jsou aplikovány na původní sejmutý obraz (*Obr.11*). Vzhledem k tomu, že v obraze jsou prakticky hledány pouze obdélníky, tak u hranových detektorů *Sobelova*, *Kirsche*, *Prewittové* a *Robinsova*, jsou využity pouze čtyři matice pro vertikální a horizontální směr. U všech metod jsou také uvedeny výsledky obrazu po *prahování*, jelikož na základě tohoto výsledku je nutné podniknout další nezbytné kroky pro jeho zdokonalení, jak už zde bylo uvedeno.

Co se týče samotných hranových detektorů, tak jak by mohla vypovídat skladba matic, výsledek *Sobelova operátoru* (*Obr.17 a*) je velice podobný výsledku *operátoru Prewittové* (*Obr.18 a*). Jediný rozdíl je v tom, že obraz po použití *Sobelova operátoru* má lehce tmavší hrany, a tak potřebuje menší prahovou hodnotu pro *threshold*. Ovšem výsledné obrazy jsou po této operaci až na pár odchylek stejné.

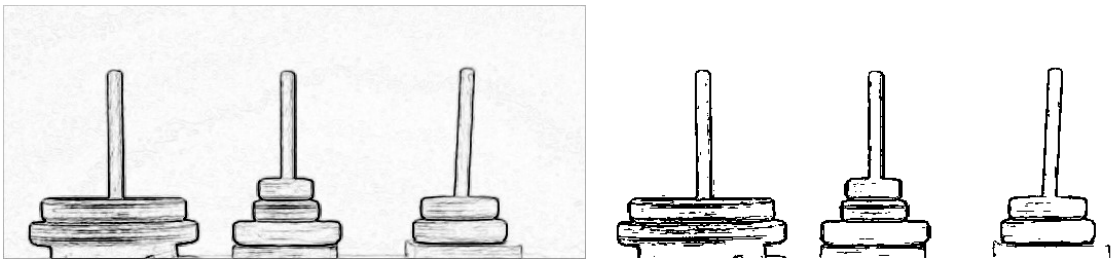


Obr. 17 a) Sobelův operátor b) Threshold s prahovou hodnotou 120

Téměř identického výsledku se dostalo i u *Robinsova operátoru* (*Obr.19 a*), jehož výstupní obraz je na první pohled nerozeznatelný od *operátoru Prewittové*, což může být díky tomu, že vzhled matice je až na středovou diagonálu, která vždy symbolizuje směr hrany, stejný. Z tohoto faktu tak lze usoudit, že zmiňovaná středová diagonála nemá u hranových operátorů příliš velký vliv.

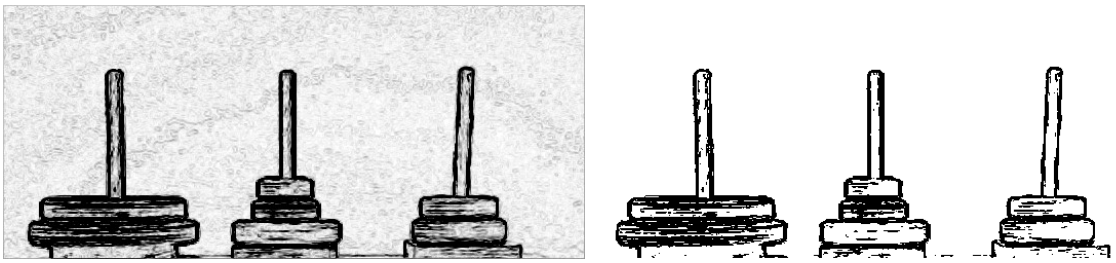


Obr. 18 a) Operátor Prewittové b) Threshold s prahovou hodnotou 165



Obr. 19 a) Robinsův operátor b) Threshold s prahovou hodnotou 170

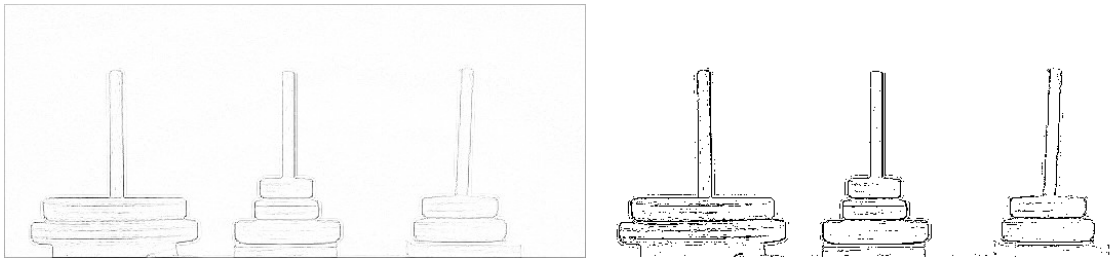
Výrazně odlišného výsledku se dostalo až u *Kirschova operátoru (Obr.20 a)*, po kterém došlo k velkému zvýraznění hran, čemuž se vzhledem k stavbě matice tohoto operátoru nedá příliš divit. Výstupní obraz je až tak výrazný, že v porovnání s ostatními operátory stačí podstatně menší prahová hodnota pro dobrý výsledek *thresholdu.*



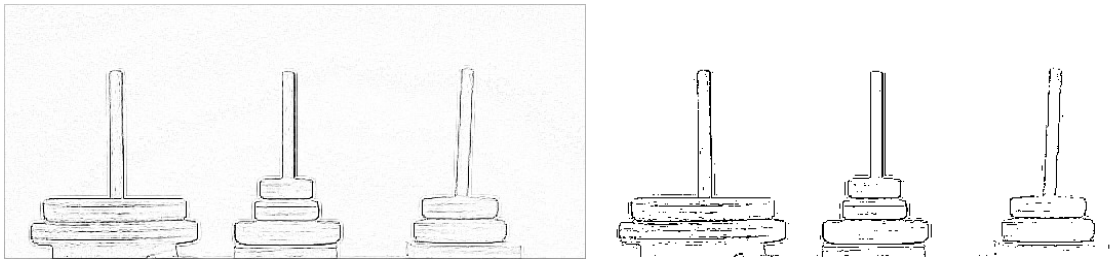
Obr. 20 a) Kirschův operátor b) Threshold s prahovou hodnotou 70

Dalšími zkoumanými hranovými detektory byly *Laplaceovy operátory*, konkrétně *Laplaceův operátor se čtyř sousedstvím* (*Obr.21 a*), *Laplaceův operátor s osmi sousedstvím* (*Obr.22 a*) a *operátor Laplacian of Gaussian* (*Obr.23 a*). U všech těchto tří operátoru si lze povšimnout patrného rozdílu oproti předchozím operátorům, kterým jsou velice tenké hrany vyskytující se v obraze. Díky těmto tenkým hranám si lze ale také všimnout nedokonalosti vstupního obrazu, jelikož se zde vyskytují dvojité hrany. Tento fakt tedy potvrzuje, že bude v konečném návrhu nutno využít některý z obrazových filtrů.

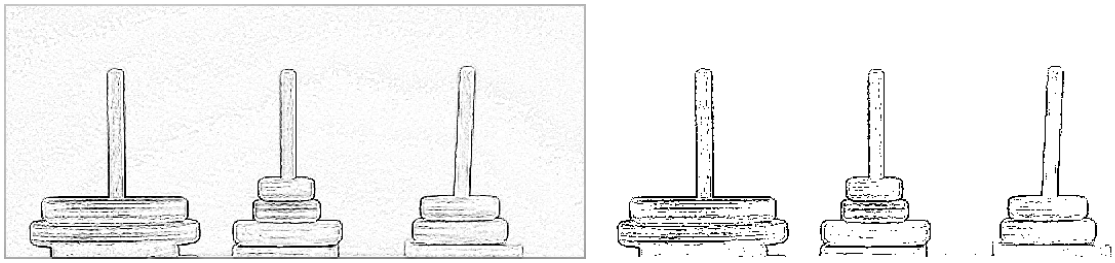
Rozdíl mezi *čtyř sousedstvím* a *osmi sousedstvím* se zdá být pouze v tom, že u *osmi sousedství* jsou hrany více tmavší, čehož lze využít u určení prahové hodnoty, jelikož není nutné detailně zkoumat která prahová hodnota je nejideálnější, protože hrany jsou už tak dostatečně oddělené. U výsledného obrazu zpracovávaného operátorem *Laplacian of Gaussian* pravděpodobně díky *Gaussovu filtru* se už nevyskytují dvojité hrany, ale na druhou stranu si lze všimnou, že veškeré objekty jsou širší a vyšší a také se zde vyskytuje více nežádaných hran na kotoučích.



Obr. 21 a) Laplaceův operátor – čtyř sousedství b) Threshold s prahovou hodnotou 235

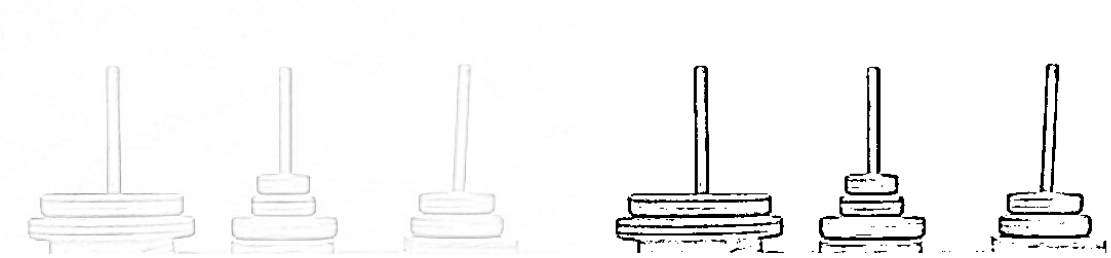


Obr. 22 a) Laplaceův operátor – osmi sousedství b) Threshold s prahovou hodnotou 190

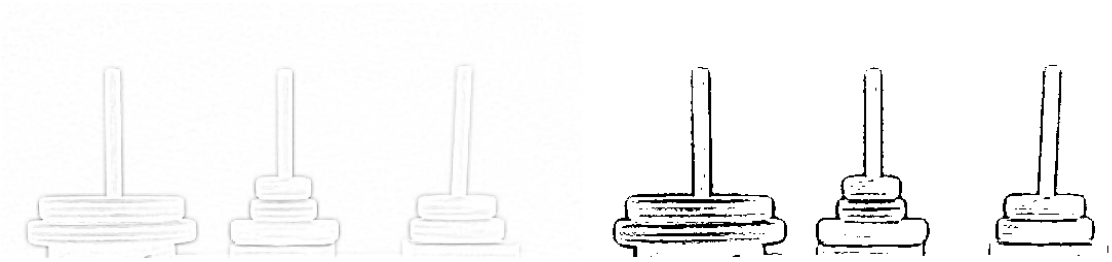


Obr. 23 a) Operátor Laplacian of Gaussian b) Threshold s prahovou hodnotou 160

Posledními zde zmíněnými hranovými detektory jsou detektory, které využívají rozmazání obrazu, jehož hodnoty jsou poté odečteny od původního nerozmazaného obrazu (*Obr.25 a*), anebo naopak je původní obraz odečten od rozmazaného (*Obr.24 a*). Na zmíněných obrázcích pro rozmazání bylo využito rozmazání *průměrováním* maticí 9x9. Co se týče situace na *Obr. 24 a,* tak výsledný obraz velice dobře zvýraznil hledané hrany, přičemž se téměř vůbec neobjevují nežádoucí hrany na kotoučích jako v předchozích případech. Jediný nedostatek je v rozích hledaných obdélníků a na některých hranách, které místy zanikají. U opačného případu (*Obr.25 a*) už není výsledek tak dobrý vzhledem k tomu, že se objevují nedostatky jako u předchozího případu, ale také se zde už opět vyskytují nežádoucí hrany a také veškeré objekty jsou širší a vyšší jako u *operátoru Laplacian of Gaussian*.



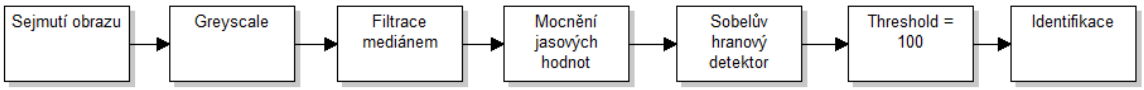
Obr. 24 a) Rozmazaný obraz mínus ostrý b) Threshold s prahovou hodnotou 245



Obr. 25 a) Ostrý obraz mínus rozmazaný b) Threshold s prahovou hodnotou 240

4.2. Návrh lokalizačního systému pro Hanojské věže

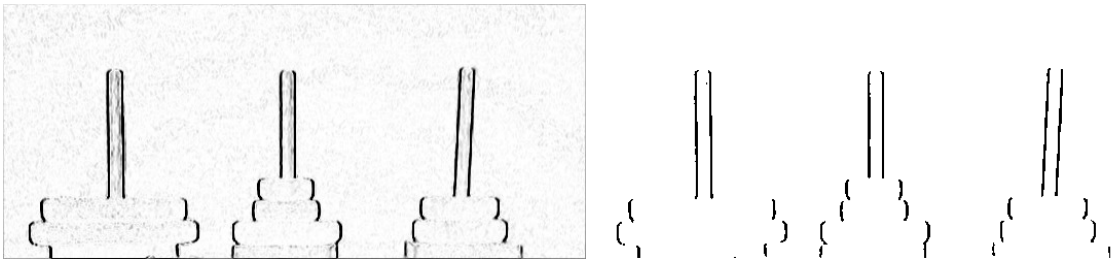
Vzhledem ke všem zjištěným faktům, jak která metoda působí na zpracovávaný obraz *Hanojských věží*, byl navržen lokalizační systém analyzující obrazový signál, jehož blokové schéma je zobrazeno na *Obr. 26*.



Obr. 26 Blokové schéma lokalizačního systému analyzujícího obrazový signál

Po sejmutí obrazu kamerou a jeho následném převedení na černobílý obraz, byla zvolena filtrace mediánem, která spolehlivě odfiltruje dvojité hrany, přičemž obraz nebude po této operaci příliš rozmazán. Dále bude následovat zlepšení kontrastu pomocí metody *mocnění jasových hodnot*, která se ukázala jako nejspolehlivější v tomto oboru. Vzhled obrazu po *mocnění jasových hodnot* je vyobrazen na *Obr. 13 b*. Po zlepšení kontrastu následuje detekce hran. Původní myšlenka, jak lokalizovat jednotlivé objekty, byla v hledání celých obdélníků, které symbolizují jednotlivé kotouče. Po testování všech hranových detektorů, které je uvedeno v kapitole 4.1.3., se však tato myšlenka ukázala jako nerealizovatelná z toho důvodu, že u jednotlivých kotoučů se výrazně projevuje i jejich konstrukce, kde se na všech jejich stranách objevují viditelné vodorovné pruhy, které hranové detektory detekují jako hrany. Poté se tedy ve výsledných obrazech po hranové detekci nenalézají celé obdélníky, ale pouze náhodné tvary, kde se dá spolehnout pouze na jejich šířku. Ale vodorovné hrany nejsou v žádném případě celistvé, některé zanikají úplně, a právě naopak se objevují nechtěné hrany zaviněné konstrukcí kotoučů. Z tohoto důvodu bylo navrhnuto využití *Sobelova hranového detektoru* pouze pro vertikální směr, jehož výsledek je uveden na *Obr. 27 a*. Z tohoto obrázku lze vidět, že horizontální hrany zcela zmizely a zůstaly pouze oblouky,

které by mohly být vhodné pro identifikaci z toho důvodu, že se z nich dá spolehlivě zjistit jak šířka, tak i výška jednotlivých kotoučů a zároveň nedošlo k zániku tyčí, na které se kotouče umisťují. Další výhodou tohoto detektoru je jednoznačné oddělení hran od zbytku obrazu, z čehož vyplývá, že není potřeba využívat automatické nalezení prahu a ani žádnou z metod matematické morfologie, což ušetří procesorový čas. Na *Obr. 27 b* je pak vidět výsledný obraz po *thresholdu*, jehož výsledek odpovídá prahovým hodnotám jasu v rozmezí od 70 do 150, takže pro práh mohla být zvolena kterákoliv hodnota z tohoto rozmezí. Takovýto obraz je pak připraven k samotné identifikaci, která bude probrána v kapitole 4.3.



Obr. 27 Výstupní obraz a) Po Sobelově hranovém detektoru – vertikální směr b) Po thresholdu s prahovou hodnotou 100

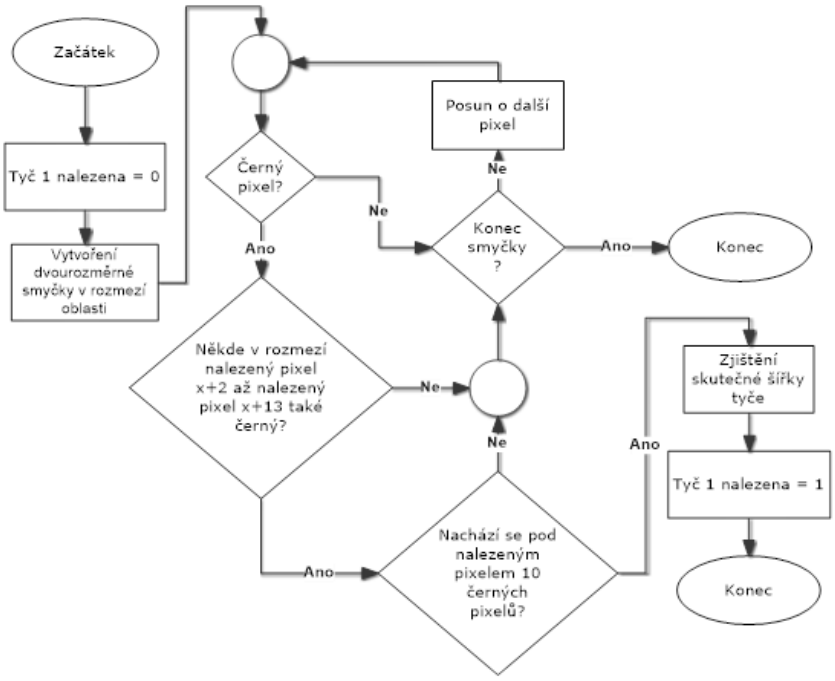
4.3. Identifikace a lokalizace objektů

Po zpracování obrazu, které je uvedeno v předchozí podkapitole, je možné se pustit do samotného rozboru nalezených hran, na jehož základě se identifikují hledané objekty. Pro dosažení cíle, kterým je lokalizování všech hledaných kotoučů byl vymyšlen postup, při kterém se nejdříve naleznou všechny tři tyče a na základě jejich pozice se určí takzvané oblasti zájmu. V těchto oblastech zájmu se poté budou identifikovat nalezené oblouky, které symbolizují hrany kotoučů. Z jednotlivých pozic a šířek kotoučů je poté možné vyhodnocovat řešení této logické hry. Zmíněný postup identifikace bude detailněji rozebrán v následujících podkapitolách.

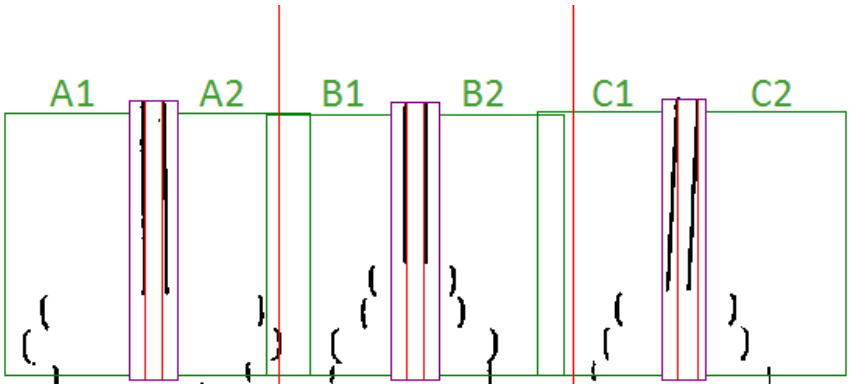
4.3.1. Lokalizace tyčí a určení oblastí zájmu

Lokalizace tyčí, na kterých jsou umístěny hledané kotouče, pracuje na tom principu, že obraz je nejdříve vertikálně rozdělen na tři stejně velké oblasti. V dalším kroku je poté v první oblasti hledána první tyč. Při tomto hledání se pomocí smyček prohledává oblast pixel po pixelu a pokud je nalezen černý pixel, tak se provádí kontroly v této oblasti tak, jak znázorňuje vývojový diagram na *Obr. 28*. Výstup hledání je tedy informace, zda byla tyč nalezena a pokud nalezena byla, tak i informace o její šířce a pozici. Po hledání v první oblasti následují naprosto stejné operace i v oblasti druhé a třetí. Po nalezení všech tří tyčí je poté určeno u každé tyče pásmo, ve kterém se nemohou vzhledem ke konstrukci nalézat žádné kotouče. Toto pásmo je zde i kvůli konstrukční nedokonalosti, kterou jsou větší otvory pro zasunutí tyčí, což má za důsledek poměrně značného vykyvování těchto tyčí, a v častých případech i nepravoúhlého postavení tyčí vůči podstavci. Kromě tohoto pásma jsou na základě pozice tyčí také určeny i jednotlivé oblasti zájmu, kde x-ová hodnota počátku je určena jako 90 pixelů od hrany tyče, y-ová jako odečtení hodnoty 8 od horního okraje tyče, šířka je k hodnotě výše zmíněného pásma a výška je téměř po dolní okraj obrazu. Výsledek funkčnosti tohoto algoritmu pro nalezení tyčí a určení oblastí zájmu je znázorněn na *Obr.*

29. V tomto obrázku jsou také označeny jednotlivé oblasti zájmu pro vysvětlení algoritmů hledání oblouků, které budou probrány v následující podkapitole.



Obr. 28 Vývojový diagram pro nalezení první tyče v obraze



Obr. 29 Nalezené tyče a určené oblasti zájmu

4.3.2. Lokalizace kotoučů

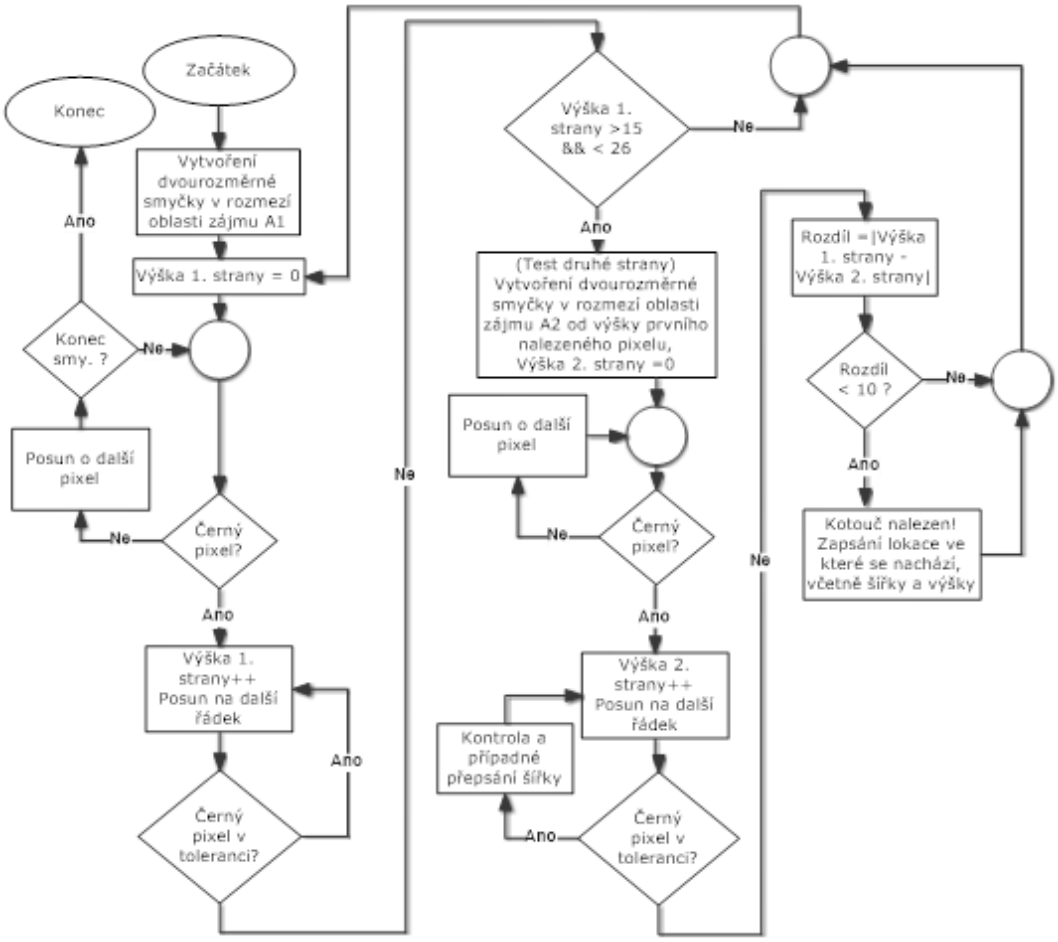
Po tom, co jsou úspěšně lokalizovány všechny tyče a jsou určeny oblasti zájmu, je možné se pustit do lokalizace kotoučů. Tato lokalizace funguje tak, že jsou nejdříve hledány kotouče na první tyči, pak na druhé, a nakonec na třetí. Princip hledání kotoučů je na každé tyči stejný, kromě pár odlišností, jako je například strana, na které se hledá první oblouk, nebo trochu složitější algoritmus u prostřední tyče, který zde musí být kvůli překrývání oblastí zájmu, a tedy možnosti výskytu oblouku v nesprávné oblasti zájmu. To musí být eliminováno správným algoritmem, který musí být schopen rozpoznat, jestli se jedná o pravý, nebo levý oblouk a na základě toho se správně zachovat.

Při hledání kotouče na první tyči je nejdříve vytvořena dvourozměrná smyčka, která nabývá hodnot souřadnic pixelů v oblasti zájmu *A1*, a tak se pomocí této smyčky dá postupně přistupovat k jednotlivým pixelům, které jsou zkoumány. Algoritmus tedy probíhá řádek po řádku, přičemž je zjišťována hodnota pixelu. Pokud je pixel bílý, tak se jde dále, a pokud je identifikován černý pixel, tak se začne zkoumat jeho okolí. Když už je tedy identifikován černý pixel, tak algoritmus už neběží po celých řádcích, ale rovnou skáče o řádek níže. Pokud je v tomto místě také černý pixel, tak se přičítá jednička k proměnné, která vyjadřuje výšku oblouku. Poté se jde zase o řádek níže a takhle dokola do té doby, než není v těchto místech už žádný černý pixel. Aby bylo zajištěno, že se skutečně jedná o oblouk ve správném směru, algoritmus do výšky oblouku 8 posouvá x-ovou hodnotu co nejvíce doleva, kde se nachází černé pixely a po této hodnotě výšky už jsou uznány pouze černé pixely v x-ové hodnotě s tolerancí 2 pixely napravo od této hodnoty.

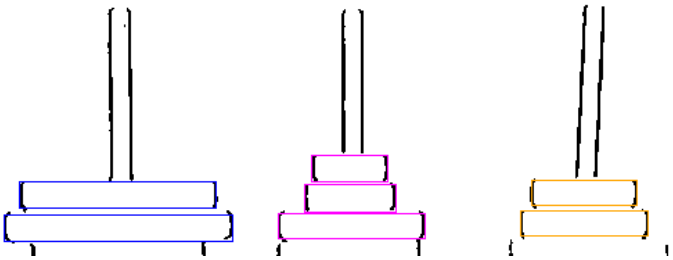
Jakmile se už níže nenachází žádný černý pixel a vysledovaná výška se pohybuje v rozmezí od 15 do 26 pixelů, tak se algoritmus přesouvá do oblasti zájmu *A2*, kde vytvoří opět dvourozměrnou smyčku, jejíž hodnoty nabývají hodnot oblasti zájmu *A2*, přičemž výška je volena v ohledu na nalezený oblouk a vždy začíná o 4 pixely níže než nejvyšší pixel levého oblouku, a to z toho důvodu, že by mohlo dojít k identifikování některé vzdálenější hrany, která patří kotouči na prostřední tyči. K tomuto jevu by teoreticky nemělo vůbec docházet, jelikož by tento potenciální špatný oblouk měl mít opačný směr, jenže praxe ukázala, že může nastat případ ve kterém boční světlo lehce rozmaže hrany a z oblouků jsou rovné čáry, které by mohly být detekovány jako správný oblouk. Rozmezí od 15 do 26 je zvoleno z toho důvodu, že hodnota menší než 15 nemůže odpovídat žádné výšce kotouče, což se dá tvrdit i o výšce nad 26, takže by se s největší pravděpodobností jednalo o hranu prstu.

Poté co se algoritmus přesunul do části *A2*, se opět hledá oblouk na stejném principu jako v oblasti *A1*, jen se hledá v opačném směru. To znamená, že poté co je detekován černý pixel, se v následujících řádcích do výšky 8 posouvá x-ová hodnota co nejvíce doprava a od hodnoty 8 se uznávají pouze černé pixely v x-ové hodnotě s tolerancí 2 pixely nalevo od této hodnoty. Poté co se nenalézá níže už žádný černý pixel se algoritmus vrací zpět do původní smyčky, kde se v absolutní hodnotě odečte výška levého oblouku od výšky pravého oblouku a pokud je tento rozdíl menší než 10, tak jsou tyto oblouky prokázány jako oblouky kotouče a je zapsána jejich lokace, výška a šířka, která je zjištěna odečtením x-ové pozice pravého oblouku od x-ové pozice levého oblouku. Na druhou stranu, pokud je rozdíl větší než 10, tak se oblouky vyhodnotí jako chyba a nic se nezapíše. Po tomto vyhodnocení se pokračuje opět v hledání černého pixelu řádek po řádku jako dříve a pokud je nějaký nalezen, tak se provádí stejné operace, jaké zde byly probrány. V případě dalšího úspěchu se nalezená lokace zapíše do proměnné symbolizující druhý nalezený kotouč na levé tyči.

Jak už bylo zmíněno, po prozkoumání oblouků v okolí levé tyče se pokračuje na zkoumání v okolí prostřední tyče, a nakonec pravé tyče, kde v každém případě pracuje podobný algoritmus. Výsledek po proběhnutí těchto algoritmů je znázorněn na *Obr. 31*. Vývojový diagram průběhu hledání kotoučů na první tyči, které zde bylo popsáno, pak znázorňuje *Obr.30*. Ještě pro doplnění, aby vůbec tyto algoritmy začaly probíhat, je nutné splnit podmínku, kterou je nalezení tyčí, a tedy i určení oblastí zájmu, ve kterých jsou oblouky hledány.



Obr. 30 Vývojový diagram lokalizace kotoučů na první tyči



Obr. 31 Výsledek lokalizace kotoučů

Lokalizace kotoučů je poslední úlohou samotné identifikace obrazu. Po této úloze už následují operace, které analyzují nalezené šířky v lokacích, na jejímž základě se vyhodnocuje řešení úlohy uživatelem. O těchto operacích však už bude zmínka v kapitole 4, která pojednává o koncové aplikaci rozhodčího pro hru *Hanojské věže*.

5. Finální aplikace

Vzhled finálního grafického rozhraní rozhodčího *Hanojských věží* je zobrazen na *Obr.32*. Před tím, než se aplikace dostane do této podoby, je po spuštění nejdříve uživatel vyzván k připojení kamery kliknutím na spodní tlačítko. Po tomto stisku si aplikace načte ze správce zařízení všechny kamery a připojí se k webkameře. Z webkamery poté v časových intervalech neustále sbírá zachycené snímky, které jsou ihned vykreslovány do příslušného okna, takže v tomto okně běží živý obraz snímaný kamerou. Pro obsluhu a práci s kamerou byly využité knihovny *Touchless.Vision.dll* a *WebCam.lib*. Po připojení kamery, které je oznámeno textem v horní části obrazovky a také spuštěním živého obrazu v okně, je uživatel vyzván k takzvané inicializaci, která je detailněji rozebrána v kapitole 5.1., ale její hlavní úkol je ten, aby uživatel seřadil všechny kotouče do výchozí polohy, která je na levé tyči. Po této inicializaci už uživatel může spustit samotnou hru a řešit tento hlavolam. Běh aplikace po spuštění hry je zase podrobně rozebrán v kapitole 5.2.

Obr. 32 Vzhled finálního grafického rozhraní pro Hanojské věže

5.1. Inicializace hry

Algoritmus, na kterém pracuje inicializace hry, je znázorněn na *Obr. 33*. Po stisku tlačítka pro start inicializace je nejdříve prováděna smyčka, ve které je zpracován obraz podle postupu znázorněného v kapitole 4.2. a následně jsou hledány všechny tři tyče v celých oblastech, jak je vysvětleno v kapitole 4.3.1. Jakmile jsou všechny tři tyče nalezeny, algoritmus přechází do další smyčky. V této smyčce je nejdříve opět zpracován obraz, ale v následujícím kroku už nejsou hledány tyče v celých oblastech jako dříve, ale pouze ve vytyčené oblasti, která je určena na základě prvního nalezení tyčí. Toto hledání v menších oblastech bylo naimplementováno ze dvou důvodů. Tím prvním hlavním důvodem bylo eliminování špatných vlivů, které do identifikace zanášely pohybující se prsty řešitele hry. Vyskytovaly se případy, kdy tyto prsty tvořily spojení jevící se jako hrany tyče někde úplně jinde, než byly skutečné tyče, a to vytvářelo značné problémy. Druhým důvodem bylo zrychlení algoritmu, který tak nemusel pro nalezení tyčí procházet tři velké oblasti, ale stačilo mu projít pouze tři menší. Pokud byly nalezeny všechny tři

tyče, tak algoritmus pokračoval k lokalizaci jednotlivých kotoučů, která je detailně popsána v kapitole 4.3.2. Po této operaci se nachází ve vývojovém diagramu dvě funkce, které v textu ještě nebyly zmíněny.

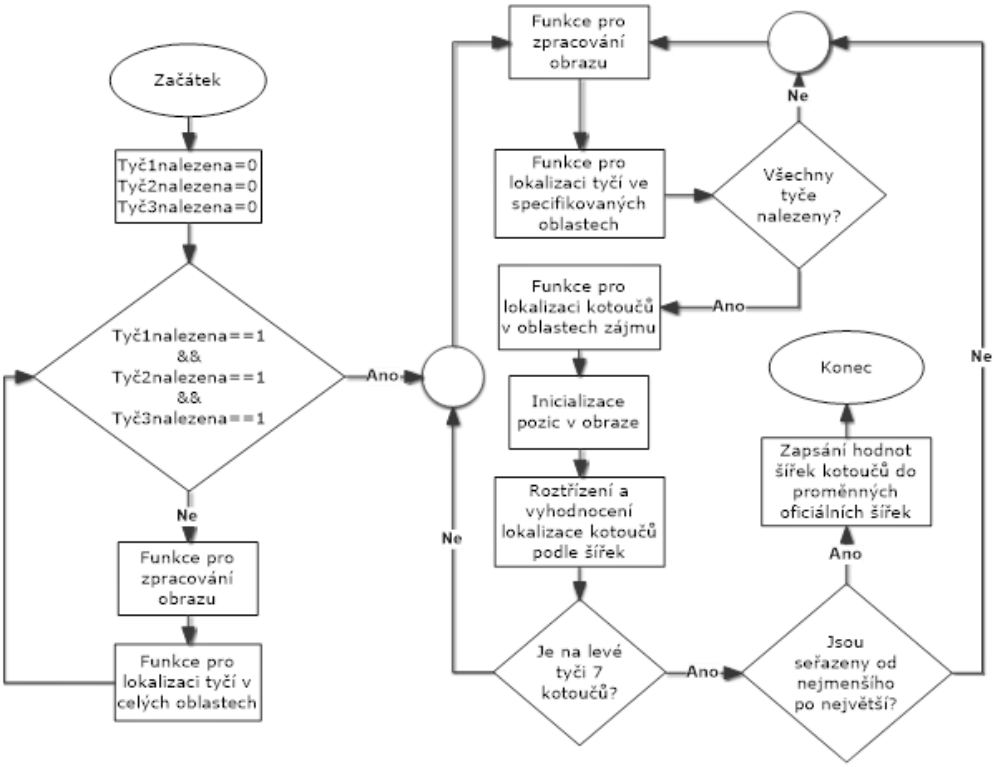
První funkcí je *inicializace pozic v obraze*. Tato funkce slouží k urovnání zjištěných šířek kotoučů na pozice přesně tak, jak jsou lokalizovány v obraze. Pro lepší vysvětlení si vezměme situaci na levé tyči v *Obr. 32*. Po průchodu algoritmem pro hledání kotoučů bude mít první nalezený kotouč, který se nachází nejvýše, pozici jedna, prostřední kotouč bude mít pozici dva a spodní kotouč pozici tři. Tato funkce díky informaci o počtu nalezených kotouč vytvoří takové výstupní hodnoty, že pozice jedna až čtyři na této tyči budou nulové, pátá pozice bude patřit šířce kotouče, který se nachází nejvýše, na šesté pozici bude šířka prostředního kotouče a na sedmé pozici bude šířka nejspodnějšího kotouče. Ve výsledku tedy tyto hodnoty budou rozmístěny podle jejich skutečné lokality.

Druhá funkce má název *roztřízení a vyhodnocení lokalizace kotoučů podle šířek*. V této funkci se nejdříve načtou všechny šířky z jednotlivých pozic do jednorozměrného pole, kde jsou poté tyto hodnoty seřazeny od nejmenšího po největší. Následně jsou takhle seřazené šířky porovnávány s původními vstupními pozicemi šířek a díky tomu je zjištěno, ve které pozici se nachází který kotouč podle velikosti. Výstup po této funkci může znázorňovat *Tab. 1* při předpokladu, že by vstupní obraz byl stejný jako na *Obr. 32*, přičemž hodnotu 7 má největší kotouč a naopak hodnotě 1 přísluší kotouč nejmenší.

Po těchto dvou funkcích už je pouze zjišťováno, jestli se na levé tyči nachází 7 kotoučů a popřípadě jestli jsou seřazeny od nejmenšího po největší. Pokud vše platí, tak se zjištěné šířky uloží do proměnných, které představují originální šířky kotoučů a inicializace je dokončena. Význam a použití těchto originální šířek bude vysvětlen v následující podkapitole. Pokud jedna z těchto dvou zmíněných podmínek neplatí, tak se podle vývojového diagramu veškeré operace provádí znovu do té doby, než jsou obě tyto podmínky splněny.

Tab. 1 Výstup funkce pro roztřízení a vyhodnocení lokalizace kotoučů podle šířek vzhledem k obrazu na Obr. 32

0	0	0
0	0	0
0	0	0
0	0	0
5	1	0
6	2	0
7	3	4



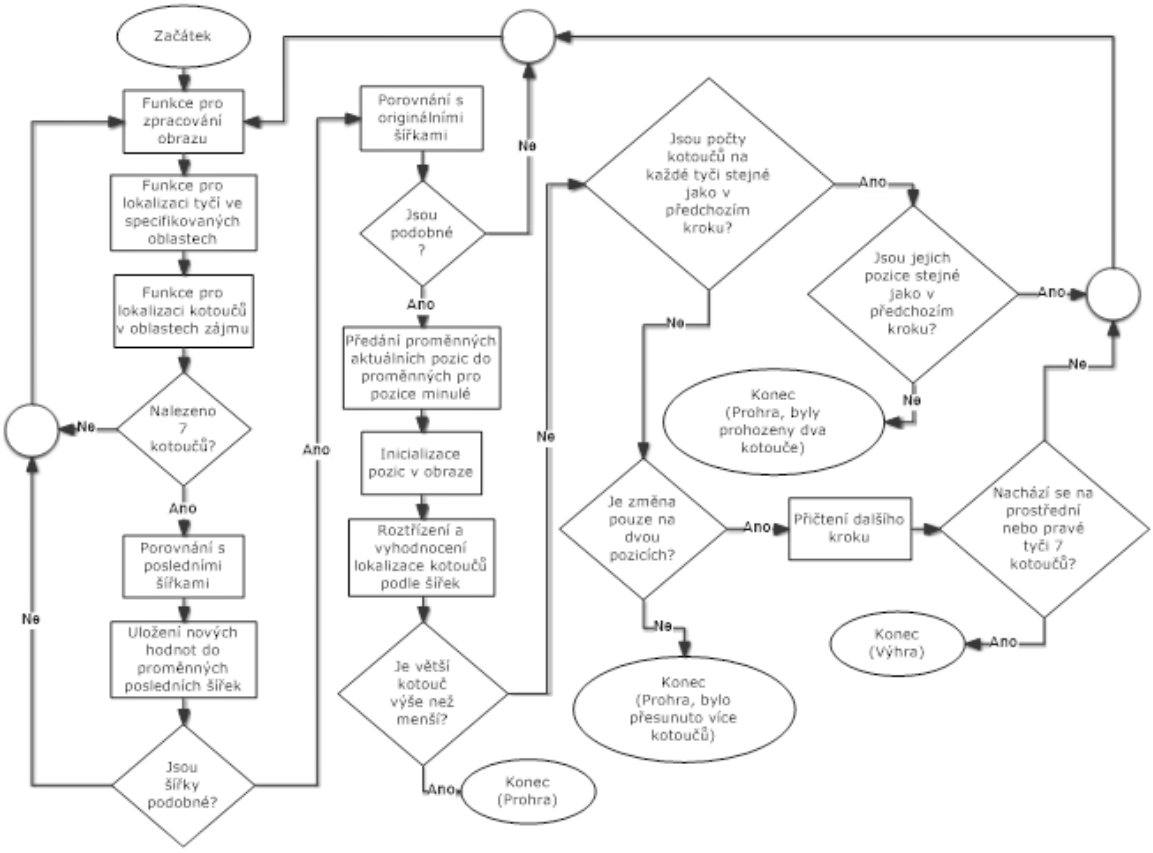
Obr. 33 Vývojový diagram průběhu inicializace hry

5.2. Průběh algoritmu rozhodčího při řešení hlavolamu

Po úspěšně provedené inicializaci je možné po stisku tlačítka spustit samotnou hru, přičemž se spustí vlákno, ve kterém běží algoritmus rozhodčího, a ještě také zároveň druhé vlákno, které se stará o počítání času. Průběh algoritmu rozhodčího při řešení hlavolamu je znázorněn na *Obr. 34*. Principiálně je po vkročení do této funkce vždy sejmut obraz, na který jsou aplikovány jednotlivé metody zpracování obrazu, popsané v kapitole 4.2. Následují funkce, které hledají tyče ve specifikovaných oblastech, kde význam této operace je vysvětlen v předchozí kapitole. Pokud jsou úspěšně nalezeny všechny tři tyče, tak se přechází k funkcím pro nalezení kotoučů v určených oblastech zájmu (kapitola 4.3.2). V dalším kroku se zjišťuje, zda bylo nalezeno všech 7 kotoučů. V případě že byl nalezen jiný počet, se algoritmus vrací zpět k funkcím pro zpracování obrazu, ale na druhou stranu v případě nalezení přesně sedmi kotoučů algoritmus přechází na další blok v pořadí.

Tímto blokem je podle vývojového diagramu *porovnání s posledními šířkami*. Tato funkce je po *funkci pro lokalizaci tyčí ve specifikovaných oblastech* druhou funkcí, která byla implementována kvůli snaze odstranit nepříznivé vlivy pohybujících se prstů. Bylo zjištěno, že pokud nastane situace, kdy dojde k chybě kvůli tomu, že algoritmy pro detekci kotoučů díky prstům naleznou kotouče i v pozicích, kde se vůbec nenachází, tak je tato situace velmi rychlá a bývá často otázkou jednoho snímku. Vzhledem k tomuto zjištěnému faktu byla vymyšlena zmíněná funkce, která ve svém těle porovnává šířky na jednotlivých lokalizovaných pozicích s šířkami, které byly zaznamenány v předchozím analyzovaném snímku. Po tomto porovnání jsou pro další analyzovaný

snímek nově zjištěné šířky zapsány do proměnných, se kterými byly porovnávány, což symbolizuje další blok v pořadí ve vývojovém diagramu. Tato funkce by tedy měla zajistit, že k pozdější analýze bude docházet pouze v případě, že bude obraz stabilní a nebudou se v něm nacházet mihotající prsty, které tak dynamicky mění nalezené šířky kotoučů. Pokud se v některé lokaci nacházela jiná šířka než v předchozím případě, tak se algoritmus vrací zpět na začátek k pořízení dalšího snímku a jeho zpracovaní. Na druhou stranu, pokud všechny šířky odpovídaly předchozím šířkám, tak algoritmus přechází k dalšímu bloku ve vývojovém diagramu.



Obr. 34 Vývojový diagram průběhu algoritmu za běhu hry

Tento blok s názvem *porovnání s originálními šířkami* je třetím a zároveň posledním blokem, který byl implementován ve snaze eliminovat nepříznivé vlivy vyskytujících se prstů. Bylo zjištěno, že při dodržení vodorovného umístění kamery s podstavcem *Hanojských věží*, mají kotouče vyskytující se na levé a pravé tyči téměř identické šířky, přičemž nezáleží, v jaké výšce se vyskytují. V případě prostřední tyče jsou šířky kotoučů podstatně menší, ale tento rozdíl je vzhledem k šířkám na pravé, nebo levé tyči konstantní. Díky tomuto zjištění tak byla naimplementována funkce, která zjišťuje podle podmínek a šířek zjištěných po inicializaci hry, zda se v obrazu nachází všechny kotouče o těchto šířkách. Pokud by tak došlo, že kvůli prstům se některý kotouč vyskytuje dvakrát, nebo i vícekrát, což by na 99 % způsobilo chybu při řešení hry, tak tato funkce to rozpozná a algoritmus se vrátí zpět k pořízení nového snímku. Pokud ovšem ale všechny šířky v určité toleranci odpovídají sedmi originální šířkám po inicializaci, tak algoritmus přejde k dalším krokům ve vývojovém diagramu.

Dalšími kroky jsou *inicializace pozic v obraze* a *roztřízení a vyhodnocení lokalizace kotoučů podle šířek*. Oba tyto kroky jsou vysvětleny v předchozí podkapitole. Než dojde k jejich provedení, tak jsou lokality kotoučů v předchozím kroku uloženy do určených proměnných, aby v následujících krocích mohl být analyzován přesun kotoučů. Po bloku *vyhodnocení lokalizace kotoučů podle šířek* jsou tedy k dispozici dvě sady proměnných, stejné jako znázorňuje *Tab.1*, kde jedna sada představuje proměnné v posledním analyzovaném kroku a druhá sada představuje proměnné v právě analyzovaném kroku. Pomocí těchto sad je možné vyhodnotit správnost uživatelova řešení.

Nejdříve dochází k vyhodnocení, zda se někde nenalézá větší kotouč než menší. Pokud ano, hra končí, ale pokud je vše správně, tak se algoritmus přesouvá k dalšímu rozhodovacímu bloku. V tomto bloku je porovnáno, zda se na všech třech tyčích nachází stejný počet kotoučů jako v předchozím kroku a na základě tohoto výsledku se algoritmus větví do dvou částí.

V první části se vyskytuje případ, že tyto počty kotoučů na jednotlivých tyčích jsou stejné, a tak jsou možné pouze dva scénáře. První je ten, že kotouče v obou sadách jsou na totožných pozicích, takže nedošlo ještě k žádnému přesunu kotouče, všechno je v pořádku a algoritmus se přesouvá zpět k pořízení snímku. Druhý scénář je takový, že uživatel se snažil prohodit dva a více kotoučů, což je chyba. Systém tedy vyhodí chybovou hlášku a ukončí hru.

Ve druhé větvi se zase vyskytuje případ, ve kterém počty kotoučů na jednotlivých tyčích neodpovídají počtům v předchozím kroku. V tomto případě je naimplementována funkce, která zjišťuje počet změn na jednotlivých pozicích oproti předchozímu kroku. Vždy když zaznamená změnu, tak k určené proměnné přičítá jedničku a pokud se na konci hodnota této proměnné rovná dvěma, tak je vše v pořádku. Uživatel provedl další krok, který se inkrementuje o jedničku na displeji a zazní tón symbolizující správné provedení kroku. Hodnota dvě odpovídá dvou změnám, kde jedna musí být z hodnoty kotouče na nulu a druhá z nuly na hodnotu kotouče. Pokud algoritmus zaznamená jinou hodnotu této proměnné, než je 2, znamená to, že uživatel přesunul více kotoučů, což je chyba. Vypíše se chybová hláška na obrazovku a ukončí se hra. Po provedení správného kroku se ještě nachází podmínka, která zjišťuje, zdali se nenachází na prostřední, nebo pravé tyči 7 kotoučů, což by znamenalo úspěšné vyřešení hlavolamu.

6. Závěr

Po zpracování a nastudování potřebné teorie k problematice ohledně zpracování a analýzy obrazu nebylo příliš jasné, jakým způsobem by mohl být obraz Hanojských věží co nejspolehlivěji a nejefektivněji zpracováván. Vzhledem k tomu tak byla nejdříve naprogramována aplikace *Testování filtrů*, ve které jsou naprogramovány jednotlivé metody pro zpracování obrazu, které mohou být poté v libovolném pořadí aplikovány na vstupní obraz. Díky tomu tak mohl být navrhnut systém, který se jevil pro pozdější identifikaci jako ten nejspolehlivější. Po zpracování obrazu byly navrhnuty algoritmy pro lokalizaci jednotlivých objektů, díky čemuž mohly být později vytvářeny rozhodovací algoritmy kamerového rozhodčího. Při lokalizaci hledaných kotoučů byl navržen postup, při kterém byly nejdříve hledány tyče, na kterých jsou umístěny kotouče. Na základě pozic těchto tyčí jsou poté určeny oblasti zájmu, ve kterých jsou hledány jednotlivé oblouky, jejichž spojení tvoří hledané kotouče.

Po tvorbě těchto algoritmů byla vytvořena finální aplikace kamerového rozhodčího, kde byly k výše zmiňovaným algoritmům doimplementovány rozhodovací algoritmy, díky kterým systém může rozpoznat, zda řešitel provedl korektní krok vzhledem k pravidlům hry, nebo se dopustil některé chyby. Na finálním grafickém rozhraní se tak objevuje snímaný obraz s vykreslenými lokalizovanými objekty, informace o počtu provedených kroků, času řešení, nebo chybové hlášky v případě chybně provedeného kroku řešitelem.

Při řešení práce se vyskytlo množství chyb, které bylo nutné vyřešit, jako například konstrukční nedokonalosti tyčí, které se v otvorech značně vykyvovaly, a tak docházelo k velkému překrývání oblastí zájmu. Dalším problémem byly vodorovné pruhy na kotoučích, které tvořily silné hrany a tak docházelo při hranové detekci k ztrácení důležitých hran, což bylo vyřešeno použitím matic pouze pro vertikální směr u hranové detekce Sobelovým operátorem. Mezi další problémy se řadilo osvětlení, které při bočním působením mazalo, nebo posouvalo boční hrany, které se začaly příliš lesknout, a tak se obraz stal neidentifikovatelným. Tento problém byl vyřešen přímým osvětlením z prostoru kamery, nebo bočním krytím okolo Hanojských věží. Posledním a zároveň největším problémem se pak staly pohybující se prsty řešitele hry, které se často jevily jako hrany kotoučů, a tak systém padal do chybných tahů řešitele hry. Tento problém byl eliminován rozmezím pro hledáním oblouků, hledáním tyčí v menších oblastech, čekáním na ustálení obrazu a porovnáním šířek kotoučů s šířkami po inicializaci.

Při dodržení zmiňovaných světelných podmínek je aplikace velice spolehlivá co se týče lokalizace jednotlivých kotoučů a následném rozhodování, zda řešitel této hry provedl správný krok, nebo se podle pravidel dopustil některé chyby. Tato bakalářská práce tak může být vhodným průvodcem pro poskytnutí informací, na jakém principu vůbec funguje zpracování obrazu, nebo jakým způsobem se dá navrhnout vlastní systém zpracovávající jakoukoliv snímanou scénu, jelikož jak už bylo v úvodu práce zmíněno, v obrazových signálech se dá rozpoznávat velká škála věcí napříč různými obory. Co se týče dalšího vývoje samotných Hanojských věží, tak by bylo snad jen vhodné vytvořit lepší konstrukci, u které by se nevyskytovaly problémy zmíněné výše, nebo by mohlo být zajímavé vyvinout robotickou ruku, která by byla schopná samostatně vyřešit tuto logickou hádanku a stát se tak neporazitelnou při rychlostních soutěžích s člověkem.

Literatura

[1] DOBEŠ, Michal. *Zpracování obrazu a algoritmy v C#*. Praha: BEN – technická literatura, 2008. ISBN 978-80-7300-233-6.

[2] ANANDAKRISHNAN, N. a S. Santhosh BABOO. *An Evaluation of Popular Edge Detection Techniques in Digital Image Processing* [online]. [cit. 2017-01-04]. DOI: 10.1109/ICICA.2014.53. ISBN 10.1109/ICICA.2014.53. Dostupné z: http://ieeexplore.ieee.org/document/6965043/

[3] SOJKA, Eduard, Jan GAURA a Michal KRUMNIKL. *Matematické základy digitálního zpracování obrazu: Matematika pro inženýry 21. století*. Ostrava, 2011. Vysoká škola báňská – Technická univerzita Ostrava a Západočeská univerzita v Plzni.

[4] HLAVÁČ, Václav a Miloš SEDLÁČEK. *Zpracování signálu a obrazu*. Praha, 1999. Elektrotechnická fakulta ČVUT v Praze.

[5] HÁJOVSKÝ, Radovan, Radka PUSTKOVÁ a František KUTÁLEK. *Zpracování obrazu v měřící a řídící technice*. Ostrava, 2010. Vysoká škola báňská – Technická univerzita Ostrava.

Seznam příloh

V přiloženém CD se nachází bakalářská práce v souboru PDF a tyto soubory se zdrojovými kódy:

- *TestovaniFiltru*
- *HanoiskaVez*