TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Webový systém pro analýzu obrazu

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie

Autor práce: Bc. Marek Jindrák
Vedoucí práce: doc. Ing. Josef Chaloupka, Ph.D.

Liberec 2017

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Web-system for image analysis

Master thesis

Study programme:	N2612 – Electrical Engineering and Informatics
Study branch:	1802T007 – Information Technology
Author:	Bc. Marek Jindrák
Supervisor:	doc. Ing. Josef Chaloupka, Ph.D.

Liberec 2017

Technická univerzita v Liberci
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2016/2017**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Marek Jindrák**
Osobní číslo:	**M15000241**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Informační technologie**
Název tématu:	**Webový systém pro analýzu obrazu**
Zadávající katedra:	**Ústav informačních technologií a elektroniky**

Zásady pro vypracování:

1. Seznamte se s problematikou zpracování a rozpoznávání obrazu.
2. Navrhněte a realizujte komplexní webový systém pro poloautomatické zpracování a rozpoznávání obrazu.
3. Tento systém by měl obsahovat graficky přívětivé prostředí, ve kterém budou použity jednotlivé algoritmy pro zpracování a rozpoznávání obrazu z knihovny OpenCV.
4. Jednotlivé funkce budou prezentovány moduly, které by měl uživatel snadno přidávat a editovat, aby bylo možné provádět i složitější analýzu obrazu.
5. Celý systém musí být navržen tak, aby byl "otevřený", tj. aby se v budoucnu daly do programu přidávat nové moduly.

Rozsah grafických prací: **Dle potřeby dokumentace**

Rozsah pracovní zprávy: **cca 40-50 stran**

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

Seznam odborné literatury:

[1] **Davies, E., R.: Machine Vision - Theory, Algorithms, Practicalities. Morgan Kaufmann Press. UK, ISBN 0-12-206093-8, 2005**

[2] **Šonka, M., Hlaváč V., Boyle. R.: Image processing, analysis, and machine vision. 3rd ed. Toronto: Thomson, 829 s. ISBN 978-0-495-08252-1, 2008**

[3] **Hlaváč, V., Sedláček, M.: Zpracování signálů a obrazů. 2. přeprac. vyd. Praha: ČVUT, 255 s. ISBN 978-80-01-03110-0, 2007**

Vedoucí diplomové práce: **doc. Ing. Josef Chaloupka, Ph.D.**
Ústav informačních technologií a elektroniky

Konzultant diplomové práce: **Ing. Karel Paleček, Ph.D.**
Ústav informačních technologií a elektroniky

Datum zadání diplomové práce: **12. září 2016**

Termín odevzdání diplomové práce: **15. května 2017**

Technická univerzita v Liberci

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

prof. Ing. Ondřej Novák, CSc.
vedoucí ústavu

V Liberci dne 12. září 2016

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum 15.5.2017

Podpis: 

Poděkování

Touto cestou bych rád poděkoval svému vedoucímu diplomové práce doc. Ing. Josefu Chaloupkovi, Ph.D. za čas a cenné rady, které mi věnoval v průběhu vytváření této diplomové práce.

Abstrakt CZ

Tato práce se zabývá návrhem a implementací systému pro analýzu a zpracování obrazu. Jsou zde vysvětleny vybrané metody určené k předzpracování, segmentaci, nalezení parametrizovatelných objektů a extrakci zájmových oblastí. Příprava potřebných dat začíná na straně klienta převedením obrazu do formátu BASE64. Následně jsou získána data z metod, které mají být aplikovány na obraz. Společně s obrazem jsou tato data přetransformována do JSON formátu, který je odeslán na server, kde je založena úloha do fronty. V odpovědi je klientské aplikaci předán identifikátor, díky kterému lze sledovat aktuální stav zpracování. Při zpracování dojde k uvolnění úlohy z fronty nezpracovaných úloh, a ta je asynchronně zpracována. Tímto způsobem byl navržen webový systém s grafickým prostředím, který umožňuje uživatelům, případně jiným systémům, zpracovávat obraz. Přínosem této práce je systém samotný, protože dává uživatelům možnost navrhnout libovolnou sekvenci podporovaných metod, které mají být na obraz aplikovány.

Klíčová slova:

Zpracování obrazu, barevný prostor, hranové detektory, segmentace obrazu, filtrace obrazu, morfologické transformace, Houghovy transformace,

Abstrakt EN

This master's thesis is dealing with project and implementation of method for analysis and image processing. It explains some methods for pre-processing, segmentation, finding of programmable objects and selection of interests sphere. A client transfers required data to BASE64 format. Obtained data are applied to the image. Data and image together are converted to JSON format which is sent to server and waiting in the queue. A client application gets data identifier, which allows to monitor activity currently. The task is during processing released from the queue of unprocessed tasks and is processed asynchronous. Web system with graphics environment was suggested this way, which allows to image processing for users or another systems. Advantage of this system is possibility to suggest any sequence of supported methods, which they are supposed to be applied to the image.

Key words:

Image processing, color space, edge detectors, image segmentation, smoothing images, morphology transformations, Hough transform.

Obsah

Seznam symbolů a zkratek

JSP	Java Server Pages
EL	Expression Language
JSTL	JSP Standard Tag Library
JSON	JavaScript Object Notation
DAO	Data access object
DTO	Data transfer object
AJAX	Asynchronous JavaScript and XML

Seznam obrázků

1 Úvod

V současné době má zpracování obrazu veliký význam a využívá se ve všech odvětvích lidské činnosti, od medicíny až po zábavu. Tento rozmach je způsoben příchodem levných přístrojů, které dokážou obrazová data ve vysoké kvalitě pořídit, ale i zpracovat. Zpracováním obrazových dat v tomto kontextu si můžeme jednoduše představit rozpoznání obličeje člověka, detekci státních poznávacích značek automobilů, ale i takzvanou rozšířenou realitu, při které je obraz analyzován a na základě nalezených kvalifikátorů i rozšířen o další data. Základem každého zpracování obrazu je jeho získání, tedy převod z analogového spojitého signálu do formy, kterou je možné zpracovávat pomocí počítače. Pojmem zpracování obrazu potom rozumíme proces, jehož vstupem je digitální obraz, sekvence obrazů a výstupem zpracovaný obraz nebo výstupní parametry, které je možno následně použít společně obrazovými daty jako vstup do dalšího procesu Průběh zpracování obrazu lze rozdělit do čtyř kroků:

- Digitalizace obrazu
- Předzpracování
- Segmentace
- Popis objektů

Jednotlivé kroky obvykle po sobě následují v uvedeném pořadí. V této práci jsou popsány základní metody každého z výše uvedeného kroku. Předzpracování obrazu je operace, která musí být provedena před jakýmkoliv zpracováním obrazu. Většina obrazů obsahuje šum nebo jiný typ rušení. Při předzpracování obrazu se obvykle využívají různé druhy filtrací za účelem odstranění šumu, operace otočení, posun, zvětšení, zmenšení, úprava jasového profilu, korekce kontrastu atd. Cílem tohoto kroku je zlepšit obrazovou informaci, která má pro zpracování největší význam. Segmentace je proces, při kterém by mělo v ideálním případě dojít k separaci jednotlivých objektů v daném obraze. Pro tyto účely se nejčastěji využívají metody prahování, detekce hran, morfologické operace atd. Typickým výstupem procesu segmentace je binární obraz. Pro popis objektů neexistuje přesný postup, který by bylo možné aplikovat na libovolný obraz, jelikož každý obraz obsahuje jiné objekty, které mohou být popsány různými barvami, umístěním, velikostí, případně svým tvarem. Výstupem tohoto kroku je seznam popsaných objektů. Na základě tohoto popisu je poté možné provést klasifikaci společných příznaků. Na vyhodnocování těchto klasifikátorů může být využita například umělá neuronová sít.

2 Využité metody zpracování a rozpoznání obrazu

V následujících sekcích jsou popsány základní principy jednotlivých metod, které jsou součástí navrženého webového systému.

2.1 Barevné modely

Barevný model popisuje základní barvy a způsob jejich míšení. Výběr vhodného barevného modelu závisí na koncovém zařízení, které s barevným prostorem pracuje. Důvodem je, že výsledné barvy se v různých barevných prostorech mohou lišit, případně některé barvy nemusí být obsaženy ve všech barevných prostorech zároveň. Barevným prostorem si pak můžeme představit množinu barev v určitém rozsahu, která je variantou některého z barevných modelů. Barevné modely lze rozdělit podle způsobu míšení barev na aditivní a subtraktivní. U aditivního míšení barev dochází k součtu jednotlivých barevných složek obrazu, a tím dochází ke zvýšení intenzity barev. U subtraktivního míchání barev dochází k rozdílu jednotlivých složek obrazu, a tím dochází ke snížení intenzity barev.

2.1.1 Model RGB

Tento barevný model je složen ze tří barevných složek (Red, Green, Blue). V případě aditivního míšení barev v plné intenzitě jednotlivých složek RGB (255, 255, 255) vzniká barva bílá. V případě, že intenzita barev je nulová RGB (0, 0, 0), tak výslednou barvou je černá. Smíšením červené a zelené složky vzniká barva žlutá. Purpurová barva vzniká smíšením červené a modré. Poslední barvou, která vznikne smíšením barev, je azurová. Ta vznikne smíšením zelené a modré složky.

Nejčastěji využívaná bitová hloubka RGB modelu je 24 bitů. Využívá se však také 32 bitová varianta, která bývá označována jako RGBA, kde A je 8 bitový kanál, nazývaný jako ALFA kanál, který udává míru průhlednosti.

Barevný model RGB bývá využíván v zařízeních určených pro zobrazování dat, např. monitor, display. Značnou nevýhodou RGB modelu je fakt, že neexistuje norma, která by přesně specifikovala vzhled základních barev. Z tohoto důvodu vzniklo několik odnoží RGB modelu, například Adobe RGB nebo sRGB

2.1.2 Model HSV

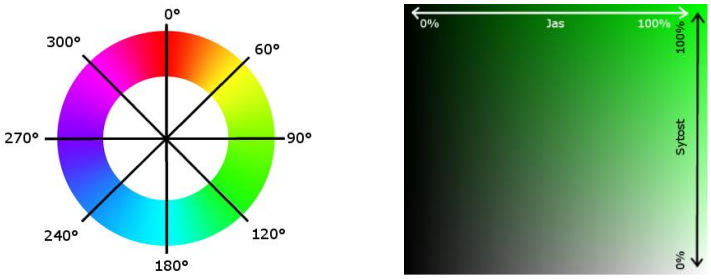
Barevný model HSV využívá pro určení výsledné barvy také tři veličiny. Na rozdíl od RGB či CMY udávají veličiny barevný odstín (*Hue*), jeho sytost (*Saturation*)

a jas (*Value*). Barevný odstín je udáván úhlem na barevném kruhu, kde každému úhlu odpovídá určitá barva. Červená barva je podle norem tvořena úhlem 0°, zelená úhlem 120° a modrá úhlem 240°. Sytost barvy udává poměr mezi základní barvou a barvou příměsi. Parametr jas udává světlost barvy. Tento barevný model je nejčastěji využíván v grafických editorech. Přínos tohoto barevného modelu spočívá v tom, že nejlépe odpovídá vnímání barev u člověka, jelikož lidské oko vnímá odstín, sytost a jas barvy. Převodní vztah je definován v rovnici (2.2)

$$H = \begin{cases} 60° \times \left(0 + \frac{G-B}{MAX(R,G,B)-MIN(R,G,B)}\right), & jestliže\ \ MAX = R \\ 60 \times \left(2 + \frac{B-R}{MAX(R,G,B)-MIN(R,G,B)}\right), & jestliže\ MAX = G \\ 60° \times \left(4 + \frac{R-G}{MAX(R,G,B)-MIN(R,G,B)}\right), & jestliže\ MAX = B \\ 0, & jestliže\ MAX - MIN = 0 \end{cases} \quad (2.2)$$

$$S = \begin{cases} 0, & jestliže \max 0 \\ 1 - \frac{MIN(R,G\,B)}{MAX(R,G,B)} & jinak \end{cases}$$

$$V = MAX(R, G, B)$$



Obrázek 1: Kruh barevných odstínů a paleta vybraného odstínu, kde na ose x je vynesen jas a na ose y sytost.

2.1.3 Barevný model YCbCr

Jde o barevný prostor, který je tvořen třemi komponenty (*Y*, *Cr*, *Cb*). Tento prostor je nejvíc využíván v oblasti digitálního videa. Výhoda tohoto modelu spočívá v kompresních vlastnostech, které vycházejí z faktu, že barevná informace je uložena pouze ve dvou komponentech (Cr, Cb). Třetí komponent udává sílu jasu (Y). Převodní vztah je definován v rovnici (2.3)

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} \begin{bmatrix} 0{,}257 & 0{,}504 & 0{,}098 \\ -0{,}148 & -0{,}291 & 0{,}439 \\ 0{,}439 & -0{,}368 & -0{,}071 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} \quad (2.3)$$

2.2 Filtrace obrazu

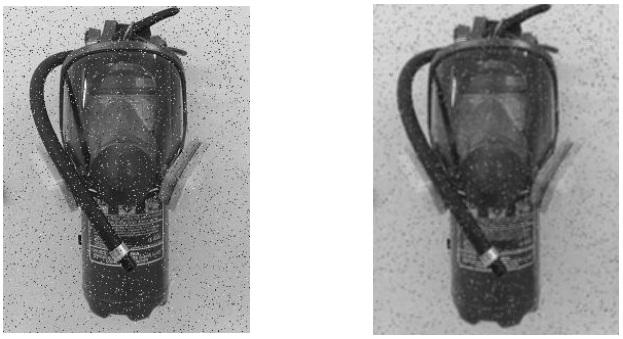
Filtrace obrazu je určena ke zvýraznění informací. Pomocí filtrace můžeme potlačit šum, vyhladit obraz nebo detekovat hrany.

2.2.1 Lineárně filtry

Intenzita bodu u lineárních filtrů je dána součtem součinů bodů v okolí a příslušnou hodnotou z konvoluční masky. Lineární flitry můžeme dále rozdělit na filtry typu dolní propust a horní propust. Nevýhodou lineárních filtrů je fakt, že jeho použitím dochází k rozmazání zpracovávaného obrazu a ztrátě jemných detailů. Součet koeficientů konvoluční masky u dolnopropustních filtrů rovná se jedna. Tyto filtry jsou vhodné pro odstraňování šumu z obrazu, jelikož šum bývá často vysokofrekvenční a filtr propouští pouze nízké frekvence. Mezi filtry dolní propust můžeme zařadit filtr typu dolní propust, který je definován vztahem (2.4)

$$g(i,j) = \frac{1}{(2E_x+1)(2E_y+1)} \sum_{k=-E_x}^{E_x} \sum_{l=-E_y}^{E_y} f(i+k, j+l)\,, \tag{2.4}$$

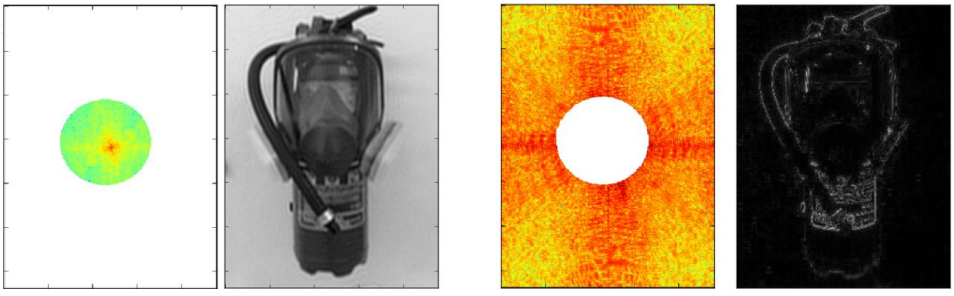
kde E_x a E_y jsou rozměry průměrovací masky, $f(i,j)$ je hodnota pixelu ve výchozím obraze, $g(i,j)$ výsledná hodnota.



Obrázek 2: Obraz před a po aplikování průměrovacího filtru

Filtry typu horní propust mají součet koeficientu konvoluční masky roven 0. Tyto filtry jsou vhodné pro zvýraznění detailů v obraze. Nevýhodou je však fakt, že spolu se zvýrazněním detailů se zvýrazní i šum.

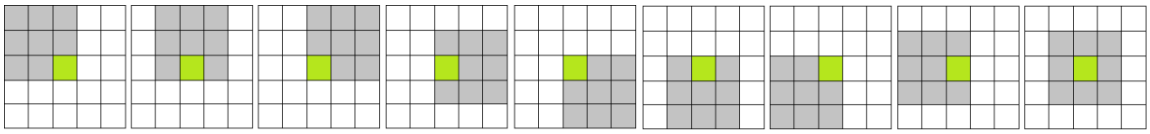
Filtraci je možné provádět i ve frekvenční oblasti obrazu viz



Obrázek 3: Z leva filtr typu dolní propust a filtr typu horní propust

2.2.2 Nelineární filtry

Intenzita bodu není vypočítávána jako v případě lineárních filtrů, nýbrž je prováděn výběr prvku z vhodného okolí. Filtr, který vyhledává minimální prvek ve vybraném z daného okolí, umožňuje potlačení šumu ve světlých částech obrazu. Naopak filtr s výběrem maximálního prvku umožňuje potlačení šumu v tmavých částech. Filtr využívající střední hodnoty je vhodný pro potlačení šumu a odstranění zrnitosti obrazu. Jeho nevýhodou je, že mění tvar hran. V případě rotující masky je velikost prohledávané oblasti 5×5. Tato oblast je rozdělena do 9 podoblastí o velikosti 3×3. U každé podoblasti je zvlášť vypočítán rozptyl jasových hodnot. Hlavní myšlenkou rotující masky je najít takovou podoblast, která má nejmenší rozptyl. Touto podoblastí je pak nahrazena vybraná oblast v obraze.



Obrázek 4: Polohy masky v rámci oblasti 5×5

2.2.3 Hranové detektory

Hrany v obraze se nacházejí v místech, kde dochází k výrazným změnám hodnot obrazové funkce $f(x, y)$. Hranové detektory nejčastěji využívají metody založené na první nebo druhé derivaci, kterou aproximují konvolucí s vhodným konvolučním jádrem. Existují však i detektory, které s derivacemi nepracují. Nevýhodou hranových detektorů založených na aproximaci derivací je jejich náchylnost k šumu.

Sobelův hranový detektor je založen na aproximaci první derivace. Výhodou tohoto přístupu je fakt, že zvýrazňuje všechny hrany nezávisle na jejich směru. Sobelův detektor umožňuje detekci hran podle směru, a to výběrem vhodného konvolučního jádra. V případě, že není kladen důraz na směr hrany, detektor provede konvoluci dané oblasti s konvolučními jádry pro všechny směry hran (2.5). Následně vybere z provedených konvolucí maximální hodnotu, kterou převede do vhodného intervalu a uloží do výsledného obrazu na dané souřadnice (x, y).

$$h_1 = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}, \quad h_2 = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix}, \quad \ldots, \quad h_8 = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{bmatrix} \qquad (2.5)$$

Laplaceův hranový detektor je založen na vlastnostech druhé derivace. Hrany nalezené pomocí tohoto detektoru se nacházejí v bodech, kde druhá derivace obrazové

funkce je rovna nule. Tyto hranové detektory bývají označeny jako „zero-crossing0“. Laplaceův hranový detektor bývá s oblibou využíván pro detekci hran, a to z důvodu, že je výrazně jednodušší hledat body obrazové funkce, kde jejich druhá derivace je rovna nule, než vyhledávat body, které dosahují maxima. Nevýhodou tohoto detektoru je fakt, že není možné získat směr hrany, jelikož jeho konvoluční jádro poskytuje identickou odezvu pro všechny směry. Mezi jeho další nevýhody patří jeho veliká citlivost na šum.

$$h_4 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \qquad h_8 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{2.6}$$

Kde h_4 je operátor pro výpočet hran z čtyř okolí a h_8 operátor pro výpočet z osmi okolí.

2.3 Geometrické transformace obrazu

Geometrická transformace je funkce, která zobrazí body x, y do $x^{'}$, $y^{'}$. Mezi jednodušší formy geometrických transformací obrazu můžeme považovat posun, který je definován vztahy (2.7)

$$\begin{aligned} x^{'} &= x + x_0 \\ y^{'} &= y + y_0 \end{aligned}, \tag{2.7}$$

kde $x^{'}$ a $y^{'}$ jsou souřadnice posunutého bodu, x a y jsou souřadnice bodu ve výchozím obrazu, hodnoty x_0 a y_0 udávají míru posunutí v daném směru.

Změna měřítka obrazu je dána vztahem (2.8). Při této transformaci dochází ke změně velikosti výsledného obrazu. Při použití této metody je také potřeba aplikovat vhodnou interpolační metodu.

$$\begin{aligned} x^{'} &= x \cdot m_x \\ y^{'} &= y \cdot m_y \end{aligned}, \tag{2.8}$$

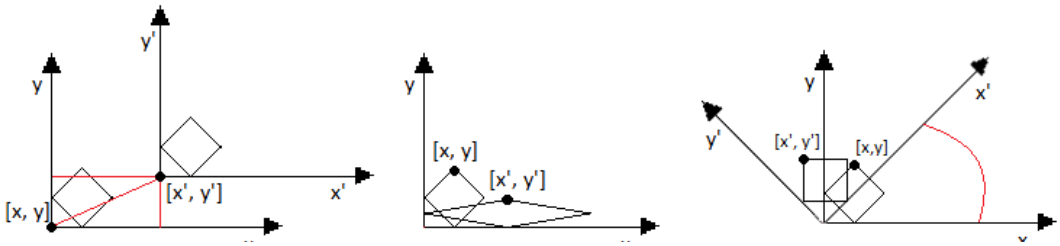
kde $x^{'}$ a $y^{'}$ jsou souřadnice výsledného bodu, x a y jsou souřadnice bodu ve výchozím obrazu, m_x a m_y vyjadřují koeficient změny měřítka v daném směru.

Mezi geometrické transformace patří také rotace, která je dána vztahy (2.9). Při rotaci však nedochází k přímému mapovaní pixelů, a je tedy nutné použít interpolaci. V případě, že rotujeme obrazem, tak můžeme zachovat původní velikost obrazu, to však

vede k ořezání některých částí obrazu. Další možností je dopočet nových rozměrů výsledného obrazu.

$$\begin{aligned} x' &= x \cdot \cos\theta + y \cdot \sin\theta \\ y' &= -x \cdot \sin\theta + y \cdot \cos\theta \end{aligned}, \tag{2.9}$$

kde x' a y' jsou souřadnice výsledného bodu, x a y jsou souřadnice bodu ve výchozím obrazu a θ je úhel natočení.



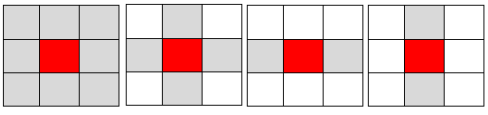
Obrázek 5: Geometrická interpretace posunu, změny měřítka, rotace

2.4 Segmentace obrazu

Segmentace je proces, při kterém dochází k redukci objemu zpracovávaných obrazových dat. Data prošlá segmentací mají užší souvislost s objekty, které se snažíme v daných datech vyhledat. Existuje několik způsobů, jak segmentaci provádět. Nejznámějším a nejjednodušším způsobem je zkoumání histogramu. Z histogramu lze vyčíst práh pro prahovací funkci. Při segmentaci obrazu můžeme využít i jiných metod, které jsou založeny na podobnosti nebo shodě bodů ve svém okolí.

2.5 Morfologické transformace

Nejčastěji jsou morfologické transformace aplikovány na binární obrazová data, ale je možné vytvořit zobecnění pro šedotónová obrazová data. Morfologickou transformaci lze chápat jako pohyb strukturního elementu v rámci daných obrazových dat. Pod pojmem strukturní element si můžeme představit libovolnou matici hodnot. Tato matice je přikládána k bodu, se kterým je proveden součet nebo rozdíl.



Obrázek 6: Strukturní elementy

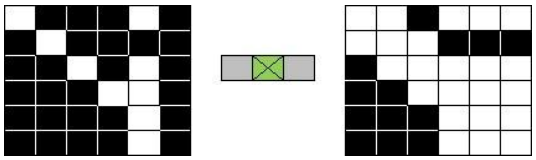
Dilatace je operace nad množinou X a strukturním prvkem B, která je rovna součtu množin. Platí tedy:

$$X \oplus B = \{x + b : x \in X, b \in B\} \tag{2.10}$$

Dilataci je také možné vyjádřit jako sjednocení posunutých bodových množin.

$$X \oplus B = \bigcup_{b \in B} (X_b) \tag{2.11}$$

Jak plyne z definice, tak je tato morfologická operace vhodná pro zaplnění malých děr v obraze, jelikož zvětšuje objekty.



Obrázek 7: Binární dilatace

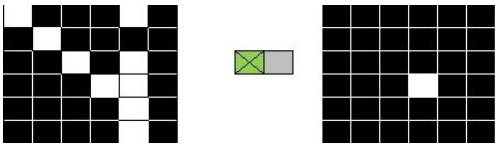
Eroze je operace nad množinou X a strukturním prvkem B, která je rovna rozdílu těchto množin. Rozdíl množin X a B je definován jako množinový doplněk součtu množinového doplňku X a strukturního prvku B.

$$X \ominus B = (X^c \oplus B)^c \tag{2.12}$$

Erozi lze vyjádřit jako průnik všech bodových posunů obrazu X o vektory $-b \in B$.

$$X \ominus B = \bigcap_{b \in B} X_{-b} \tag{2.13}$$

Operace eroze není inverzní funkcí k dilataci, nýbrž je operací duální. Eroze bývá využívána pro odstranění objektů v obraze, které jsou menší než strukturní element. Pomocí eroze lze také rozdělit objekt na menší objekty.

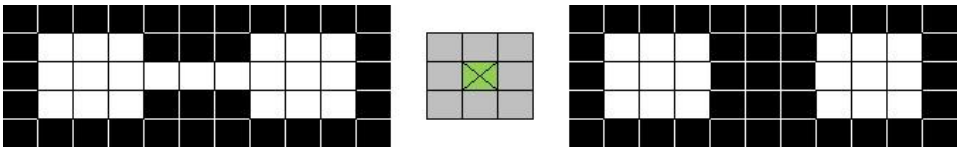


Obrázek 8: Binární eroze

Morfologické otevření je operace eroze, po které následuje operace dilatace.

$$X \circ B = (X \ominus B) \oplus B \tag{2.14}$$

Výhodou je, že operace otevření nemění původní rozměr a tvar. Tato operace bývá často využívána pro odstranění šumu nebo k odstranění tenkých linií mezi objekty.

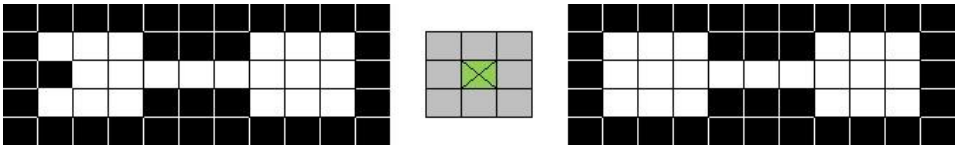


Obrázek 9: Binární otevření

Morfologické uzavření je operace dilatace, po které následuje operace eroze.

$$X \cdot B = (X \oplus B)\ \Theta B \tag{2.15}$$

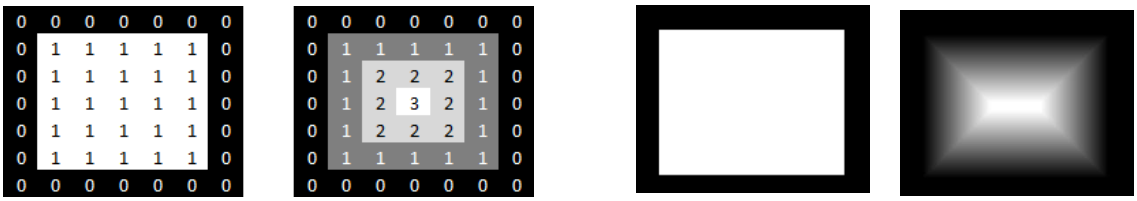
Výhodou je, že operace otevření nemění původní rozměr a tvar. Tato operace je využívána ke spojení blízkých objektů případně k zaplnění malých děr.



Obrázek 10: Binární uzavření

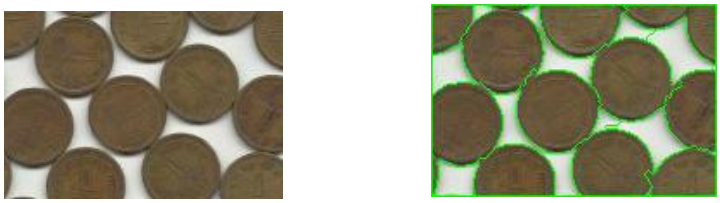
Vzdálenostní funkce je morfologická operace s binárním obrazem, kde nuly lze chápat jako pozadí a jedničky odpovídají segmentovaným objektům (popředí). Vzdálenostní funkce je závislá na zvolené metrice, kde vzdálenost na šachovnici odpovídá strukturnímu elementu o velkosti 3×3, křížový strukturní element odpovídá vzdálenosti městských bloků a Euklidovská vzdálenost odpovídá kruhovému strukturnímu elementu. Nejjednodušší, ale také neefektivní způsob, jak tuto transformaci provést, je vykonat několikanásobnou erozi s vybraným strukturním elementem, kdy v každém kroku označíme patřičnou hodnotou odloučenou část. Efektivnější je dvouprůchodový algoritmus výpočtu vzdálenostní transformace. Pomocí tohoto algoritmu je možné snadno a efektivně vypočítat vzdálenostní transformaci pro metriky - vzdálenost na šachovnici a vzdálenost městských bloků. Euklidovskou vzdálenost pomocí tohoto algoritmu není jednoduché na dva průchody vypočítat. Můžeme však zavést takzvanou kvazieuklidovskou aproximaci (2.16), pro kterou tento algoritmus lze snadno použít.

$$D_{QE}\big((i,j)(h,k)\big)\begin{cases}|i-h| + \left(\sqrt{2}-1\right)|j-k| & pro \quad |i-h| > |j-k| \\ \left(\sqrt{2}-1\right)|i-h| + |j-k| & jinak\end{cases} \tag{2.16}$$



Obrázek 11: Obraz před a po provedení vzdálenostní funkce

Transformace rozvodím je morfologická transformace, kde je obraz typicky chápán jako reliéf krajiny. Jde o postupné zaplavování oblastí vodou. Zaplavování probíhá do doby, než je dosaženo nejvyššího bodu v terénu. Tam, kde se střetnou tyto oblasti, vznikají hráze. Tyto hráze rozdělují obraz na jednotlivé segmenty.



Obrázek 12: Originální obraz a obraz s vyznačenými hranicemi.

2.6 Histogram

Je nejjednodušší pomůcka využívaná při zpracování obrazu. Histogram sumarizuje počty jednotlivých úrovní šedi daného obrazu. Každý obraz má jen jeden histogram, který se však může shodovat s histogramem jiného obrazu. Formálně lze histogram definovat jako funkci (2.17)

$$N = \sum_{i=0}^{n} h(i) \tag{2.17}$$

kde n udává počet úrovní a $h(i)$ je počet jasových úrovní i.

Ekvalizace histogramu je proces, při kterém dochází ke zvýšení kontrastu obrazu. Jasové intenzity jsou přepočítávány tak, aby byly zastoupeny v co nejširším rozmezí. Hodnoty jasu spočteme jako funkci (2.18)

$$N = \frac{J_{max}}{x * y} \sum_{i=J_{min}}^{i=J_{max}} h(i) \tag{2.18}$$

kde J_{max} je maximální a J_{min} je minimální hodnota jasu ve výchozím obraze, x a y jsou rozměry výchozího obrazu, $h(i)$ je počet pixelu odpovídající této hodnotě.

Obrázek 13: Obraz před a po ekvaliazcí histogramu

2.7 Fourierova transformace

Fourierova transformace obecně slouží k převodu signálu z časové oblasti do oblasti frekvenční a je definovaná integrálem (2.19) nebo diskrétním popisem pomocí sum.

$$F(\varepsilon) = \int_{-\infty}^{\infty} f(t) e^{-2\pi i \varepsilon t}\, dt \tag{2.19}$$

$$F(k) = \frac{1}{N} \sum_{n=0}^{N-1} f(n)\ exp(-2\pi i \frac{nk}{N}) \tag{2.20}$$

kde ε je frekvence a $2\pi\varepsilon$ je úhlová frekvence, $F(k)$ je spektrum s periodou N a $f(n)$ je diskrétní signál. Inverzní Fourierova transformace je F^{-1} je definována jako

$$f(t) = \int_{-\infty}^{\infty} F(\varepsilon) e^{2\pi i \varepsilon t}\, d\varepsilon \tag{2.21}$$

$$f(n) = \sum_{n=0}^{N-1} F(k)\ exp(2\pi i \frac{nk}{N}) \tag{2.22}$$

Jelikož obraz je dvojrozměrný signál, tak lze jednoduše ze znalosti (2.19) odvodit Fourierovu transformaci pro dvojrozměrné signály (2.23), a to i v diskrétní variantě (2.24).

$$F(u,v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x,y) e^{-2\pi i (xu+yv)} du\, dv \tag{2.23}$$

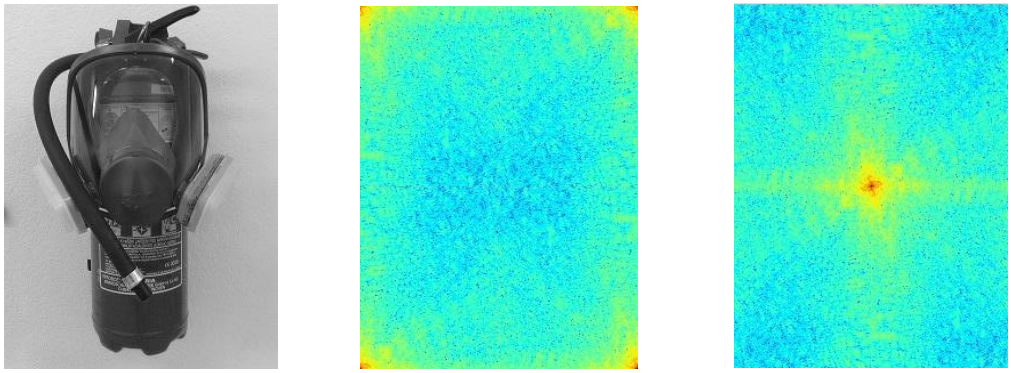
$$F(u,v) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m,n)\ exp\left[-2\pi i \left(\frac{mu}{M} + \frac{nv}{N}\right)\right] \tag{2.24}$$

Z rovnice (2.25) a (2.26) pak můžeme odvodit inverzní 2D Fourierovu transformaci.

$$f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(u,v)e^{2\pi i\,(xu+yv)}du\,dv \tag{2.25}$$

$$f(m,n) = \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)\;exp\left[2\pi i\left(\frac{mu}{M}+\frac{nv}{N}\right)\right] \tag{2.26}$$

Aplikací 2D diskrétní Fourierovi transformace na obraz lze získat necentralizované spektrum. To spektrum má nízké frekvence v rozích. Na základě symetrií spektra je možné spektrum rozdělit na kvadranty a ty přeskupit tak, aby byly nízké frekvence uprostřed spektra.



Obrázek 14: Originální obraz, jeho zlogaritmované amplitudové spektrum, přeskládané amplitudové spektrum

2.8 Identifikace objektů

Identifikace objektů je proces, při kterém jsou v zadaných datech vyhledávané zájmové objekty. Mezi tyto objekty můžeme zařadit přímky, kružnice, případě celé matice hodnot.

2.8.1 Houghova transformace

Klasická Houghova transformace je určena k detekci parametrizovatelných objektů, jako například přímek nebo kružnic, ale je možné tuto transformaci zobecnit i pro složitější objekty.

Při vyhledávání přímek lze použít popisu přímky, která je definována rovnicí (2.27)

$$y = kx + q \tag{2.27}$$

V rámci využití Hougových transformací je však vhodnější využít normálový tvar (2.28)

$$y = \frac{x \cdot \cos(\gamma)}{\sin(\gamma)} + \frac{r}{\sin(\gamma)}, \tag{2.28}$$

kde r je vzdálenost přímky od počátku a úhel γ je v intervalu $\langle 0, 2\pi \rangle$ a je popsán úhlem, který svírá osa x s normálovým vektorem. Pro všechny přímky, které protínají bod $[x, y]$, vypočteme parametr r a zapíšeme jej do Houghova prostoru jako část harmonické funkce. Vyhledávaná přímka je pak dána maximy v Houghově prostoru.

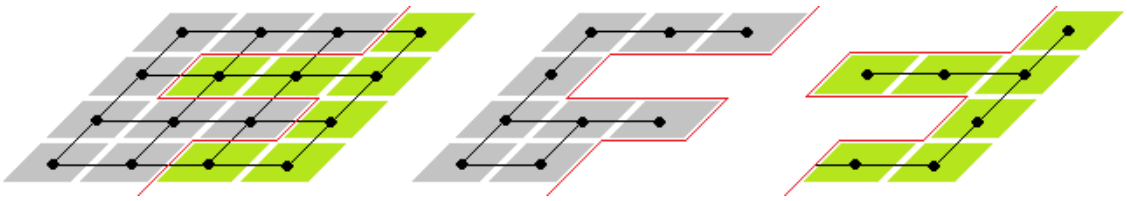
Při vyhledávání kružnic lze postupovat obdobně jako v případě vyhledání přímek, zde využijeme obecnou rovnici pro popis kružnice (2.29)

$$r^2 = (x - x_0)^2 + (y - y_0)^2, \tag{2.29}$$

kde r je poloměr kružnice, bod $[x_0, y_0]$ je středem kružnice a x, y jsou souřadnice bodu v obraze. Pro každý hranový bod měníme hodnoty x_0, y_0 a dopočítáváme r. Z toho plyne, že všechny body, které nejsou hranové, jsou potencionálními středy hledané kružnice. V případě, že se tento potencionální střed kružnice vyskytuje v Houghově prostoru často, tak se nepochybně v obraze vyskytuje kružnice s poloměrem r a středem kružnice v bodě $[x_0, y_0]$.

2.8.2 GrapCut

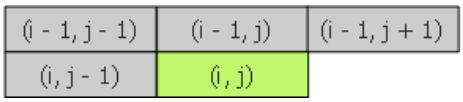
Je metoda založená na grafových algoritmech, které provádějí řez grafem. V prvním kroku této metody musí být vybráno několik pixelů, které reprezentují hledaný objekt. Takto vybrané pixely jsou označovány jako inicializační body. Tyto body jsou pak vždy součástí hledaného objektu, a to ať už v popředí či pozadí obrazu. Při této operaci je vypočítáno globální optimum ze všech operací, které odpovídají zadání, které je definováno inicializačními body. Typicky bývá označena obdélníková část v obraze, kde se nachází popředí, tedy hledaný objekt. Tuto oblast obvykle lze ještě rozšířit o masku, která rozšiřuje inicializační body, díky čemuž lze poměrně přesně označit celé zájmové objekty.



Obrázek 15: Vlevo je uveden vstupní obraz, ve kterém je červeně zvýrazněn provedený řez grafu, obraz uprostřed znázorňuje pozadí a obraz vpravo popředí po provedení řezu.

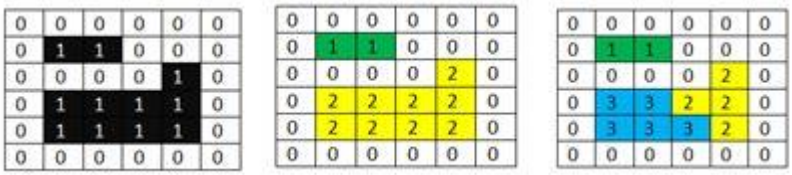
2.8.3 Barvení oblastí

Metoda barvení oblastí je založena na vyhledávání maximálních skupin sousedících pixelů v binárních obrazových datech.



Obrázek 16 Prohledávaná oblast

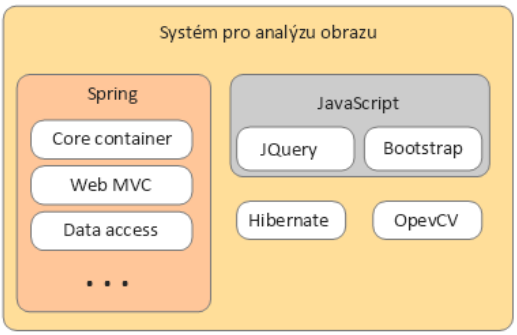
Obrazová data jsou vždy prohledávána zleva doprava a od shora dolů. Každá skupina sousedících pixelů je označena unikátním číslem (barvou), které ovšem nemusí znamenat pořadí barvy. V případě, že dojde při prohledávání ke kolizi barev, tak je tento fakt zaznamenán. Druhým průchodem jsou všechny sousedící oblasti přebarveny stejnou barvou.



Obrázek 17: Vstupní binární obraz, první průchod, druhý průchod.

3 Návrh systému pro analýzu systému

Tato část práce je věnována popisu systému a technologiím, které byly v rámci diplomové práce využity. Byla využita řada technologií, které umožňují vytvoření systému s předem zadanými podmínkami, mezi které můžeme zařadit snadnou rozšiřitelnost, přívětivé uživatelské prostředí a v neposlední řadě také knihovny, které musí být nutně v rámci projektu využity.



Obrázek 18: Architektura systému

3.1 Framework Spring

Jedná se o framework, díky kterému lze snadno vyvíjet enterprise aplikace v jazyce Java. Jeho hlavní výhodou je fakt, že aplikace nemusí běžet na aplikačním serveru, jelikož Spring je na něm nezávislý a lze jej tedy použít i pro vývoj standardních aplikací. Další velkou výhodou je snadná konfigurace, která může být definována XML souborem přímo v kódu pomocí anotací, nebo je možné tyto dva přístupy zkombinovat.

Samotný Spring si zakládá na myšlenkách obráceného řízení (Inversion of Control - IoC) a vkládání závislostí (Dependency Injection - DI). IoC je návrhový vzor, díky kterému je možné uvolnit těsné vztahy mezi jednotlivými komponenty aplikace. Těsnou vazbu mezi objekty si je možno představit jako třídu A, která využívá další třídu B. V případě využití IoC si třída A nevytváří sama instanci třídy B, ale je jí poskytnuta jiným způsobem. Způsobů, díky kterým lze poskytnout instanci B je několik, například pomocí injekcí konstruktorem nebo pomocí settru. DI je pak konkrétní technikou IoC.

3.2 REST

REST (Representational State Transfer) je architektura, díky které je možné pomocí základních HTTP metod přistupovat k datům. Základní metody jsou vytvoření (POST), přijetí (GET), aktualizace (PUT) a smazání (DELETE). Data přenášena na

webovou službu jsou ve formátu JSON. Pomocí tohoto pak lze asynchronně komunikovat na server, a tak měnit data v pozadí při vykonávání uživatelské akce.

3.3 Hibernate

Hibernate je framework, který umožňuje objektově relační mapování (tzv. ORM). ORM je proces, při kterém dochází k překladu objektu na tabulkovou reprezentaci, současně s tím jsou přeloženy i vztahy mezi jednotlivými objekty, které obvykle bývají uloženy v dalších tabulkách

3.4 Knihovna OpenCV

OpenCV je open source knihovna zaměřena na počítačové zpracování obrazu. Tato knihovna je velmi rozsáhlá a poskytuje nástroje vhodné na filtrování obrazu, segmentaci obrazu, detekci objektů, transformaci objektů, ale i práci s videem. Tuto knihovnu lze snadno využít ve spoustě programovacích jazyků C, C++, Python nebo Java. Její výhodou je tedy multiplatformost a fakt, že je tato knihovna stále rozvíjena.

3.5 JQuery a Bootstrap

JQuery je javascriptová knihovna, která umožňuje snazší práci javaskriptem. Její hlavní výhodou je jednoduchost, rychlost a multiplatformost. Pomocí této knihovny lze například:

- vytvářet efekty, animace
- snadno manipulovat s objekty modelu dokumentu (DOM)
- modifikovat kaskádové styly
- spravovat události
- získávat data ze serveru bez potřeby obnovy celé stránky (AJAX)

Bootstrap je open source knihovna, která obsahuje velké množství nástrojů určených pro tvorbu webových aplikací. Tato knihovna je podporována velkou většinou současných i minulých verzí webových prohlížečů. Díky této knihovně je také možné dynamicky měnit obsah právě zobrazované stránky, a to v návaznosti na aktuálně zvoleném rozlišení.

3.6 Uživatelské prostředí

Při návrhu uživatelského prostředí bylo využito značkovacího jazyku HTML, který byl vhodně rozšířen o styly, které definují design a chování celé stránky.

Obrázek 19: Grafické rozhraní

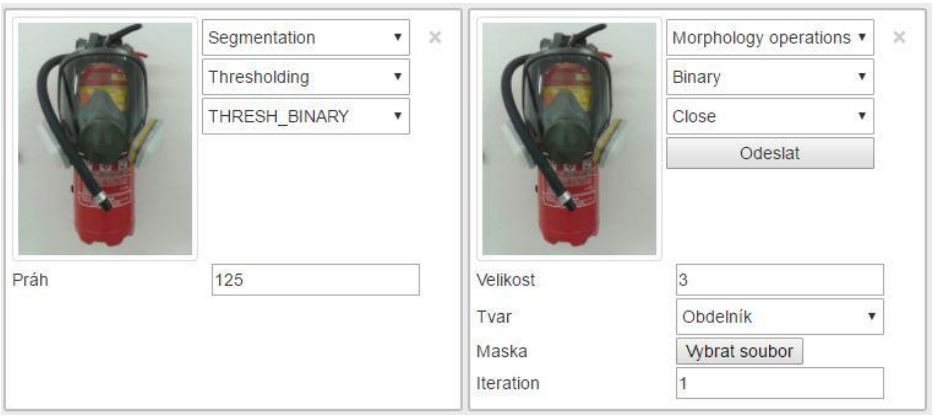
Uživatelské prostředí systému je díky využití framworku Bootstrap responsivní, a je rozděleno do čtyř základních stránek. První stránka je domovská stránka systému, na kterou uživatel přijde nejprve. Odtud může uživatel pohodlně vstoupit na druhou stránku, kde probíhá interakce systému s uživatelem. Na další stránku byl umístěn seznam všech podporovaných metod, které jsou v systému implementovány. Pro každou metodu je zde uveden seznam metod, které jí smí předcházet. V rámci této práce je nejdůležitější stránka, která je určena pro interakci uživatele se systémem. Ve výchozím stavu jsou na této stránce umístěny dvě dlaždice. První dlaždice je určena pro nahrání vstupního obrazu.

Při nahrávání souboru je v pozadí provedena konverze obrazu do formátu BASE64. Po provedení této konverze jsou data dočasně uložena ve skrytém elementu,

který náleží této dlaždici. Současně s tím proběhne změna výchozího obrázku dlaždice. Druhá dlaždice je určena pro přidání dalších dlaždic, které definují sekvenci úloh, které mají být s vloženým obrazem provedeny. Každá takto přidaná dlaždice obsahuje výchozí obrázek, tři select boxy (funkce, metoda, operace), a sekci pro odstranění vybrané dlaždice. Data do select boxu určeného pro funkce jsou načítána asynchronně hned při zobrazení stránky. V závislosti na vybrané funkci jsou pak načteny metody. Po výběru metody jsou načteny operace, které je možno s danou metodou provést. Současně s výběrem operace jsou načítány její atributy, pomocí kterých lze konfigurovat jednotlivé operace. Každá operace může disponovat různým počtem a typem atributů.

Atributy mohou být trojího typu:

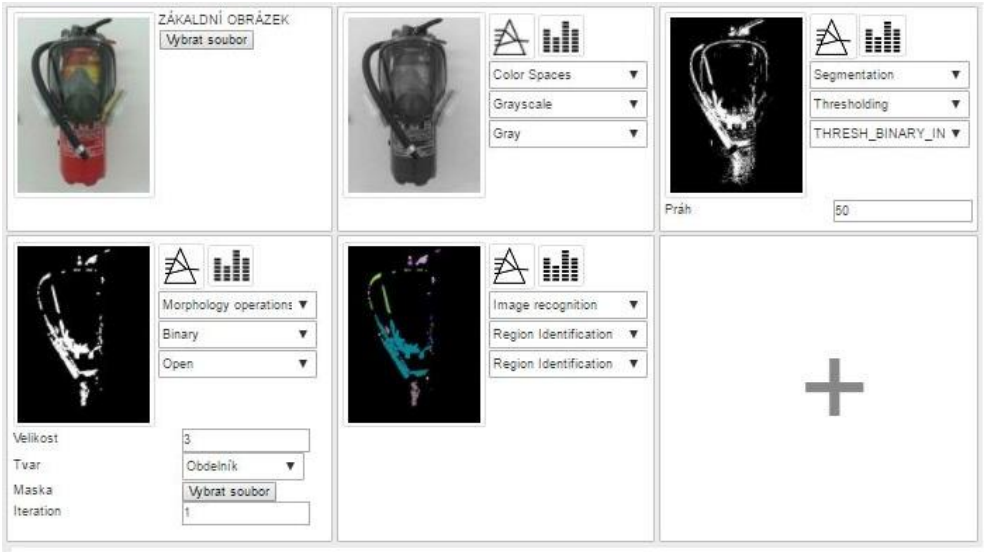
- Hodnota
- Výčtový typ
- Obrázek



Obrázek 21: Dlaždice s různým typem a počtem atributů

Výška všech dlaždic je závislá na maximálním počtu atributů v jednotlivých dlaždicích. Tlačítko odeslat je vždy vykresleno na dlaždici, která byla jako poslední upravena. V případě kliknutí na toto tlačítko jsou vykonány tři akce. První akcí, která je vykonána, je načtení dat ze všech dlaždic a převedení do JSON formátu. V druhé akci jsou data odeslána na serverovou část, kde je provedena kontrola dat a následností jednotlivých metod. V případě, že je vše v pořádku, je vygenerován unikátní klíč, pomocí kterého je možné sledovat aktuální stav zpracování požadavku a uživateli je zobrazeno upozornění o úspěšném založení požadavku na zpracování. Poté je spuštěna třetí akce a její prací je získávání aktuálního stavu zpracování právě vytvořeného

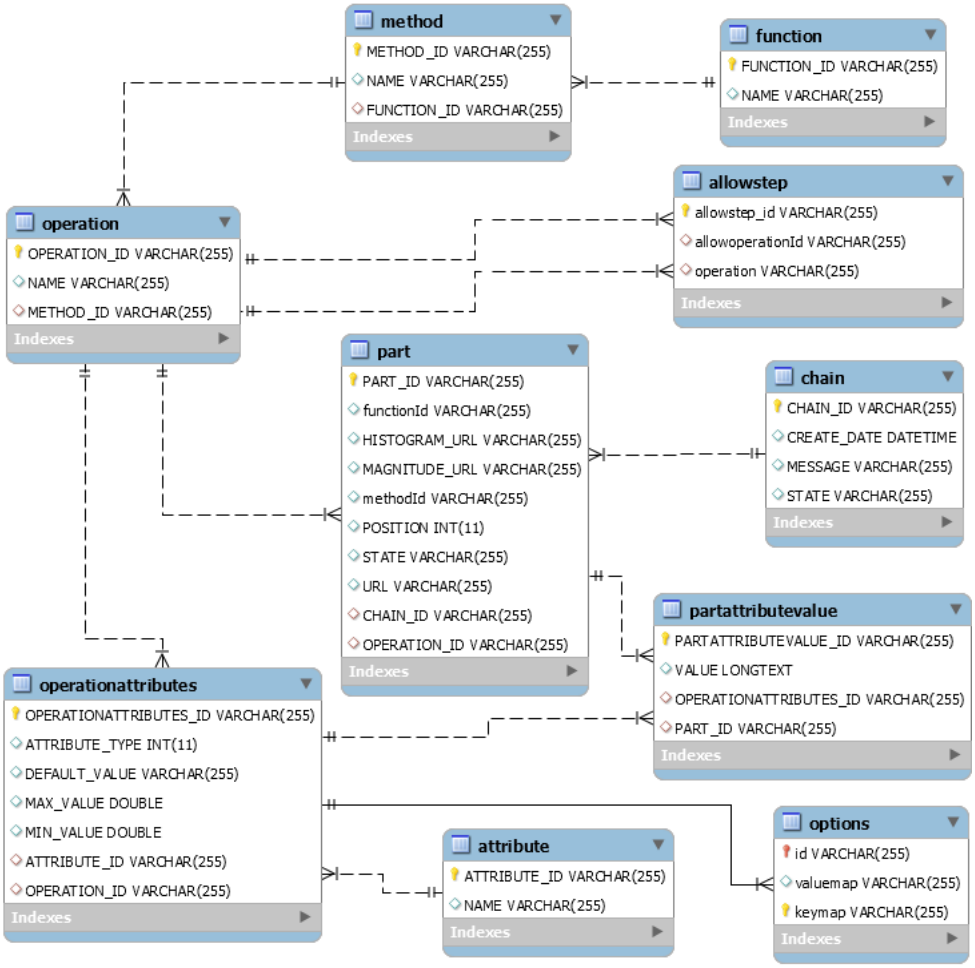
požadavku. Tato fáze může být ukončena třemi způsoby. Prvním způsobem je úspěšné zobrazení zpracovaného požadavku. Druhý způsob ukončení je vyvolán ze strany serveru, a to z důvodu výskytu chyby v průběhu zpracování požadavku. Třetím způsobem je vytvoření nového požadavku.



Obrázek 22: Ukázka zpracované sekvence operací

3.7 Návrh databázového modelu

Pro správnou funkčnost celého systému byl navržen datový model, který obsahuje deset entit a pokrývá všechny požadavky, které byly na model kladeny. Základní požadavek, kladený na model, byl, aby bylo možné pod jednu funkci zařadit metody a pod jednotlivé metody operace. Tento požadavek byl zajištěn pomocí entit functions, method, operation a relacemi mezi nimi. Ke každé operaci by mělo být také možné přiřadit libovolný počet atributů, u kterých by mělo být možné nastavit minimální, maximální a výchozí hodnotu v případě, že se jedná o číselný typ. V případě, že se jedná o výčtový typ, by mělo být možné nastavit seznam možností. Tento požadavek byl zajištěn pomocí entit operation, operationAttributes, attribute, options a relacemi mezi nimi. Dále bylo požadováno, aby bylo možné vytvořit řetěz operací, které mají být sekvenčně vykonány. Tomuto požadavku bylo vyhověno vytvořením entit chain, part, operation, partAttributes a relacemi mezi nimi. Posledním požadavkem bylo umožnit vyhodnocování následností jednotlivých operací. Za tímto účelem byly vytvořeny entity allowstep, operation a relace mezi nimi.



Obrázek 23: Databázový model systému

3.7.1 Konfigurace Hibernatu mapování

Konfigurace Hibernatu byla provedena ve třídě AppliactionConfig. Důležité je zde správně nastavit DataSource. Nastavení v tomto případě spočívá v nastavení driveru, který má být použit, URL kde je umístěn databázový server, jména a hesla pro přihlášení do databáze. URL může obsahovat další informace, například zda má být databáze vytvořena v případě, že neexistuje, zda se má použít SSL, maximální povolenou velikost paketů atd. Dalším důležitým krokem je nastavení SessionFactory. Zde musí být nutně zaregistrovány všechny třídy, které reprezentují jednotlivé databázové entity.

3.7.2 Objektově relační mapování

Navržený databázový model bylo zapotřebí převést do objektů tak, aby s entitami modelu bylo možné snadno pracovat v javě. Pro tento účel bylo využito frameworku Hibernate. Aby tato konverze byla možná, musela být pro každou entitu vytvořena třída, která ji reprezentuje. Každá takováto třída musí být opatřena anotacemi

@Entity a @Table(name =“Table_name“) a v rámci této práce musí implementovat rozhraní CustomEntity. Každá proměnná této třídy musí být opatřena anotací @Column(name=“col_name“). Proměnná, která reprezentuje primární klíč, musí být opatřena anotací @Id. Relace jsou zde řešeny pomocí anotací @ManyToOne, @JoinColumn a @OneToMany

```
@Entity
@Table(name = "method")
public class Method implements Serializable, CustomEntity {
    @Id
    @Column(name = "METHOD_ID")
    private String methodId;

    @Column(name = "NAME")
    private String name;


    @OneToMany(mappedBy = "method")
    private Set<Operation> operations = new HashSet<Operation>();

    @ManyToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "FUNCTION_ID")
    private Function function;
//GETTERS
//SETTERS
}
```

Zdrojový kód 1: Ukázka mapování entity method na objekt Method

3.7.3 Využití rozhraní CustomEntity

Jak již bylo řečeno, každá třída, reprezentující entitu, musí implementovat rozhraní CustomEntity. Na základě toho rozhraní bylo totiž možné vytvořit abstraktní vrstvu, která je určená pro repositáře. Tato abstraktní vrstva definuje základní operace, které je možné s těmito objekty provádět a její implementace se nachází v abstraktní třídě BasicRepositoryAbstract. Většina metod v této abstraktní vrstvě je opatřena anotací @Transactional. Každá tato metoda totiž musí být zpracována transakčně. V případě, že metoda proběhne korektním způsobem, je automaticky zavolán commit, v opačném případu proběhne rollback.

```
@Transactional
public <T extends CustomEntity> void save(T entity) {
    sessionFactory.getCurrentSession().save(entity);}
```

Zdrojový kód 2: Ukázka transakční metody, která uloží libovolnou entitu splňující rozhraní CustomEntity

3.7.4 Inicializace databáze a registru operací

Tento systém je navržen tak, aby se při startu zinicializoval do výchozího stavu, který je definován ve třídě DbProvisioner. Tato třída je opatřena anotací @Component a implementuje rozhraní InitializingBean. Při startu Spring inicializuje jednotlivé komponenty a v rámci této inicializace je automaticky volána metoda afterPropertiesSet, která je v této třídě přepsána. Před samotným zavoláním této metody jsou zinicializovány všechny objekty učené pro práci s entitami, takzvané „Data access object“ dále jen DAO. Každý DAO je rozšířen abstraktní vrstvou BasicRepositoryAbstract a poskytuje tedy základní operace. Dále každý DAO obsahuje specifické metody, které jsou pevně svázány s entitou, kterou zastupují.

Při zavolání metody afterPropertiesSet je nejprve vytvořen registr operací, který je definován třídou OperationRegister. Jedná se o třídu, která dodržuje návrhový vzor singleton, a lze tedy vytvořit pouze jedinou instanci této třídy. V tomto registru jsou zaregistrovány všechny operace, které systém umí zpracovat. Každý záznam v tomto registru obsahuje informaci o operaci, třídě, která jí umí zpracovat, a klasifikátoru, který přesně vymezuje akci, která bude systémem vykonána. Následuje inicializace a uložení jednotlivých tříd, které zastupují entity. Třída zaregistrovaná v tomto registru musí nutně splňovat rozhraní IMethodWorker

```
operationRegistr.register(redOperation.getOperationId(),
RGBChannel.class, ChannelsEnum.RED.getChannelName());
```

Zdrojový kód 3: Ukázka registrace operací

3.8 Návrh systému

Tento systém byl navržen podle návrhového vzoru Model-View-Controller (dále jen MVC). MVC je v současnosti nejvyužívanějším návrhovým vzorem použitým při návrhu webových aplikací. Jeho základní myšlenkou je oddělení prezentační vrstvy od logiky systému. Tento návrhový vzor se skládá se ze tří základních částí:

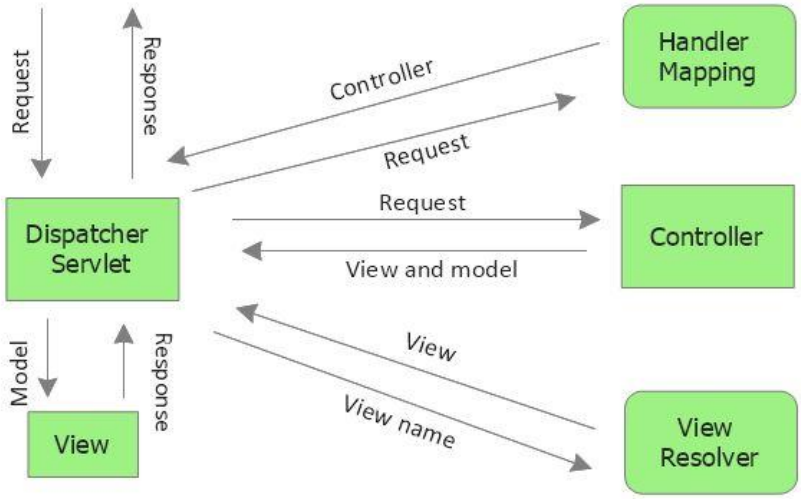
- Model – obsahuje data aplikace, která mají být předána pohledu. Tato data obvykle bývají získávána z databáze.
- View(pohled) – V případě webových aplikací je výstupem HTML kód, který je podle předem dané šablony doplněn o data z modelu.
- Contoller (kontroler) – Umožňuje uživateli provádět požadované akce, na základě kterých je aktualizován obsah modelu. V případě provedené

změny v modelu je kontaktován pohled, který zajistí přegenerování výstupu.

Obrázek 24: Obecný model návrhového vzoru MVC

Framework Spring disponuje vlastní implementací návrhového vzoru MVC tzv. Spring Web MVC. Obalem tohoto mechanismu je třída DispatcherServlet, jejímž účelem je příjem http požadavků od uživatele. Obsahuje aplikační a servletový kontext. Aplikační kontext se stará o životní cyklus komponent servisní a datové vrstvy. Servletový kontext má na starosti komponenty spojené web MVC.

Vyřízení požadavku je závislé na dvou základních komponentech HandlerMapping a ViewResolwer. Úkolem HandlerMappinguje je určit, která metoda kontroleru má být vykonána. Úkol ViewResolveru sestává ze tří částí. První část je zodpovědná za nalezení odpovídajícího pohledu, druhá za vložení modelu do pohledu a poslední část je zodpovědná za vygenerování odpovědi. Definice vlastního kontroleru je prováděna pomocí anotace @Controller. Tímto způsobem je aplikačnímu kontextu předán objekt, který má být zavedený jako beana při skenování komponentů. Aby bylo možné rozlišit metody kontroleru, je nutné definovat cestu. Ta je definována anotací @RequestMapping, která jí přijímá jako textovou hodnotu. Dalšími atributy této anotace je možné specifikovat metodu požadavku, na kterou má reagovat (GET, POST, …)



Obrázek 25: Zpracování požadavku Spring MVC

3.8.1 Prezentační vrstva

Zobrazení prezentační vrstvy systému obhospodařuje třída BasicController. Tato třída je opatřena anotací @Controller a obhospodařuje pouze požadavky typu GET:

- / - Zobrazí uživateli view, které reprezentuje domovskou stránku. V tomto případě dojde k zobrazení obsahu uloženého v souboru index.jsp
- **/application** - Zobrazí uživateli view, které reprezentuje stránku, ve které je umístěna aplikace. V tomto případě dojde k zobrazení obsahu uloženého v souboru application.jsp
- **/workAssignment** - Zobrazí uživateli view, které reprezentuje stránku, ve které je uvedeno zadání této práce. V tomto případě dojde k zobrazení obsahu uloženého v souboru workAssignment.jsp
- **/devdoc** - Zobrazí uživateli view, které reprezentuje stránku s dokumentací. V tomto případě dojde k zobrazení obsahu uloženého v souboru devdoc.jsp

V metodách, které obhospodařují tyto požadavky, můžeme do modelu vložit libovolná data, která lze poté využít v jednotlivých view.

```
@Controller
public class BasicController {

  @Autowired
  private ContentProviderService contentProviderService;

  @RequestMapping(value = "/devdoc", method = RequestMethod.GET)
  public String devDoc(Model model) {
    List<AllowStepsDTO> as = contentProviderService.getAllowSteps();
    model.addAttribute("allowSteps", as);
   return ViewConst.DEVDOC;
  }
…
{
```

Zdrojový kód 4: Ukázka vlastního kontroleru obhospodařující požadavky na cestě /devdoc

3.8.2 Práce s daty

Příjem a poskytování dat obhospodařuje třída JsonReceive, která je opatřena anotacemi @Controller a @RequestMapping("/rest"). Tento kontroler poslouchá

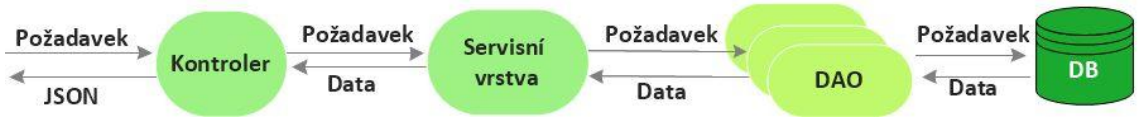
požadavky jdoucí na /rest. Všechny metody této třídy poskytují data ve formu JSON. Metody této třídy obhospodařují následující požadavky:

- **/rest/createChain** – Tato metoda je vykonána při přijetí požadavku typu POST. V těle tohoto požadavku musí být data o řetězci operací, které mají být s obrazem provedeny. V prvním kroku této metody je kontaktována servisní vrstva, která vyhodnotí, zdali je posloupnost operací validní či nikoliv. V případě, že je posloupnost kroků validní, je kontaktována servisní vrstva, která podle dat v těle požadavku založí úlohu na vykonání požadovaných kroků s obrazem. Při úspěšném založení je kontroleru vrácen identifikátor, podle kterého je možné zpracování sledovat. Tento identifikátor je pak spolu s dalšími daty vložen do odpovědi a odeslán zpět tazateli. V případě, že je zadaná nevalidní sekvence operací, je na tuto skutečnost uživatel upozorněn.
- **/rest/isChainReady/{chainId}** – Tato metoda reaguje na požadavek typu GET. Nejprve je z cesty vyňat identifikátor řetězu operací a hned poté je kontaktována servisní vrstva, která zajistí potřebné informace. Na základě získaných informací ze servisní vrstvy je poté rozhodnuto o tom, jaká data jsou vrácena tazateli. V podstatě mohou nastat tři situace
  1. Sekvence operací byla úspěšně zpracovaná
  2. Zpracování sekvence ještě nebylo dokončeno
  3. V průběhu zpracování došlo k chybě

  Součástí každé odpovědi je
  1. Zpráva pro uživatele
  2. Seznam operací a jejich dat
  3. Informace o tom, zda je zpracování dokončeno
  4. Informace o výskytu chyb v průběhu zpracování
- **/rest/getFunctions** – Tato metoda je spuštěna přijetím požadavku typu GET. V prvním kroku je kontaktována servisní vrstva, která požadavek zpracuje a vrátí kontroleru seznam všech funkcí, které systém umí zpracovat. Tento seznam je poté vložen do těla odpovědi.
- **/rest/getMethod/{functionId}** - Metoda je vykonána po přijetí požadavku typu GET. Nejprve je získán identifikátor funkce, který je poté předán servisní vrstvě. Servisní vrstva na základě obdrženého

identifikátoru poskytne seznam metod dané funkce. Tento seznam metod je vložen do odpovědi a odeslán zpět žadateli.

- **/rest/getOperation/{methodId}** – Při přijetí požadavku typu GET je spuštěna metoda, ve které nejprve dojde k separaci identifikátoru funkce, a poté je zkontaktována servisní služba, které je tento identifikátor předán. Servisní služba zajistí, aby byla kontroleru vrácena požadovaná data. Tato data jsou následně vložena do odpovědi a vrácena zpět tazateli.
- **/rest/getAttributes/{operationId}/{pageAttributeId}** – Metoda je vykonána po přijetí požadavku typu GET. Nejprve jsou vyparsována data z cesty a poté jsou předána servisní vrstvě, která je zodpovědná za získání seznamu atributů. Tento seznam je vložen do odpovědi a zaslán zpět tazateli.



Obrázek 26: Průběh zpracování dat

3.8.3 Datová vrstva

Datová vrstva je určena pro přístup k datům. K tomu účelu slouží třídy, které jsou opatřeny anotací @Repository. Tyto objekty bývají často označeny jako DAO. Jak již bylo řečeno v jedné z předešlých sekcí, každé DAO je rozšířeno základní abstraktní vrstvou BasicRepositoryAbstract, která umožňuje provádět základní operace s entitou, kterou zastupuje. Každé DAO také může obsahovat své specifické metody určené pro práci s entitou. Těchto objektů bylo v rámci práce navrženo devět AllowStepDAO, AtributeDAO, ChainDao, FunctionDao, MethodDao, OperationAttributeDao, OperationDao, PartAtributeDao, PartDao.

Data, která jsou získána z datové vrstvy, by neměla být dále distribuována do vyšších vrstev, protože se jedná o entitní objekty. Ty by neměly projít přes servisní vrstvu. Toto lze zajistit již zde, v datové nebo až servisní vrstvě, převedením do přepravních objektů, takzvaných „Data transfer object“, dále jen DTO. Do těchto objektů mají být vkládána pouze data, která mají význam pro vyšší vrstvy a nevkládají se do nich celé entitní objekty, ale pouze jejich hodnoty.

3.8.4 Servisní vrstva

Servisní vrstva je zodpovědná za přípravu dat, která jsou vyžadována kontrolery. V rámci této práce byly vytvořeny dvě třídy, které tuto vrstvu zapouzdřují. Tyto třídy jsou opatřeny anotací @Service a za jejich inicializace je odpovědný Spring.

První servisní vrstvu zastupuje třída ChainValidator. Tato třída poskytuje metody, které jsou určeny pro validaci vstupních dat od uživatele. Vstupními daty je zde myšlen seznam operací, které mají být s obrazem vykonány. Prvním validovaným atributem jsou obrazová data, která jsou zde stále ještě ve formátu BASE64. V případě, že tento atribut zde není nalezen, servisní vrstva vyvolá výjimku ImageNotFoundException. Tuto výjimku předá nadřízenému objektu, tedy kontroleru, a ten odešle uživateli oznámení, že tuto sekvenci není možné zpracovat, jelikož nebyla vložena obrazová data. Dále je validována vzájemná následnost jednotlivých operací. V případě, že je nalezena nevalidní následnost operací, je tento proces zastaven a sekvence operací je označena jako nevalidní. Vyhodnocení této skutečnosti je pak už závislé na kontroleru.

Druhou a komplexnější servisní vrstvu zapouzdřuje třída ContentProviderService. V této třídě jsou umístěny všechny metody, které umí zpracovat data do podoby, která je vyžadována kontrolery. Jsou zde popsány metody, servisní vrstvy:

- getAllFunctions – Úkolem této metody je získání všech podporovaných funkcí. K těmto datům přistupuje přes datovou vrstvu, která poskytuje potřebná data. Tato datová vrstva je zapouzdřena třídou MethodDAO. Data z této vrstvy jsou převedena do seznamu objektu typu ListDataDTO, který je vyžadován v místě, kde byla volána tato metoda servisní vrstvy.
- createWholeChain – Tato metoda je zodpovědná za založení validní sekvence operací, které mají být aplikovány na obraz. Vstupními daty této metody je seznam operací. Tato metoda je komplexnější a využívá několik datových vrstev. Nejprve vytvoří samotný řetěz, ke kterému připojuje části, které jsou definované vstupními daty. Výstupem je identifikátor této sekvence operací.

- isChainReady – Vstupním argumentem této metody je identifikátor sekvence, na základě kterého jsou vyhledávaná potřebná data. Výstupem této funkce je pak seznam jednotlivých operací. Tato metoda poskytuje všechna aktuální data týkající se této sekvence, a to z důvodu, aby bylo možné uživateli aktuálně zobrazovat stav zpracování.

3.8.5 Zpracování sekvence operací

Zpracování sekvence operací bylo navrženo tak, aby se provádělo automaticky v předem definovaných intervalech. K tomuto účelu byla vytvořena třída AsyncTaskExecutor, která má pomocí anotace @EnableAsynch povolené asynchronní zpracování. V této třídě je dále implementována metoda execute, které je pomocí anotace @Scheduled s parametrem fixedDelay nastaven interval, jak často má být spuštěna. Před prvním spuštěním této metody je do kontextu této třídy vložena Springem instance třídy ThreadPoolTaskExecutor. Způsob, jakým má být tato třída vytvořena, je uveden v konfigurační třídě Appconfig. V rámci této práce může vzniknout pouze jedna instance třídy ThreadPoolTaskExecutor. To je dáno implementací metody getAsyncExecutor z rozhraní AsyncConfigurer. Tato metoda definuje, jakým způsobem má být instance třídy ThreadPoolTaskExecutor vytvořena. Lze zde například nastavit:

- název prefixu vlákna,
- maximální velikost zásobníku vláken,
- velikost hlavního zásobníku vláken,
- kapacitu fronty.

Po spuštění metody execute nejprve dojde k zjištění stavu zásobníku vláken. V případě, že je tento zásobník plný, se neprovede žádná akce a vyčká se do příštího spuštění, kdy se provede stejná validace. V opačném případě je z datové vrstvy načten seznam nejstarších nezpracovaných sekvencí operací. Poté proběhne změna stavu právě načtených sekvencí. Je jim nastaven stav „Processing“. Pro každou sekvenci je poté založena úloha, která je předána exekutoru k vykonání. Každá tato úloha musí nutně splňovat rozhraní Runnable, a musí tedy implementovat metodu run. V této metodě je spuštěno samotné zpracování vytvořením instance třídy Workflow a jeho spuštěním pomocí metody run. V prvním kroku tohoto zpracování je nastaven stav všech částí sekvence na „Processing“ a poté je započat průchod sekvencí. V prvním průchodu je

vytvořena datová struktura, do které je ukládán aktuální výsledek zpracování, a která je zároveň použita jako vstup do dalšího kola zpracování. V každém kroku průchodu sekvencí je volána metoda processStep, která zapouzdřuje zpracování obrazu vybranou operací. Tato metoda jako přijímá jako vstupní data obraz, na který má být aplikována vybraná operace, data o vybrané operaci a informaci o tom, zdali se jedná o první průchod touto metodou.

```
private void startWorkFlow() {
    logger.info("Workflow strated");
    BufferedImage data = null;
    boolean firstStep = true;
    for (Part part : sortedParts) {
        try {

            data = processStep(data, part, firstStep);
        } catch (NoDataFound noDataFound) {
            noDataFound.printStackTrace();
        } catch (SelectionLayerException e) {
            e.printStackTrace();
        }
        firstStep = false; }}
```

Zdrojový kód 5: Ukázka odbavení částí sekvence operací

V metodě processStep nejprve dojde k vyhodnocení, zda se jedná o první průběh touto metodou. Na základě toho je rozhodnuto, která data jsou vložena konstruktoru třídy WorkflowStep při vytváření její instance. Následně jsou z této instance načtena zpracovaná data, která jsou použita jako výstupní data z této metody. V případě, že je průběhu vykonání operace vyvolána výjimka¨, je odchytávaná v této metodě. V takovém případě je nastaven chybový stav, chybová hláška a proces vykonání je zastaven.

Implementace třídy WorkflowStep umožňuje automatický výběr třídy, která je schopna operaci zpracovat. Tato třída musí implementovat rozhraní IMethodWorker, které definuje základní metody, které lze využít. Výběr vhodné třídy a vytvoření její instance je proveden na základě identifikátoru aktuálně zpracovávané části sekvence. Tento identifikátor je předán tovární metodě třídy MethodFactory. Ta si nejprve získá instanci třídy OperationRegister a potom z tohoto registru přečte název třídy, která má být vytvořena a vytvoří ji. Následně nastaví klasifikátor, který je v pozdějších krocích využit pro rozlišení operace, která má být vykonána. To nastavení musí proběhnout z důvodu, že tato třída může obhospodařovat více operací a jediný způsob, jak je rozdělit, je využít tento klasifikátor. Tato instance je poté vrácena zpět, kde je do ní vložen zbytek potřebných dat. Poté je spuštěno vykonání metodou work. Po provedení

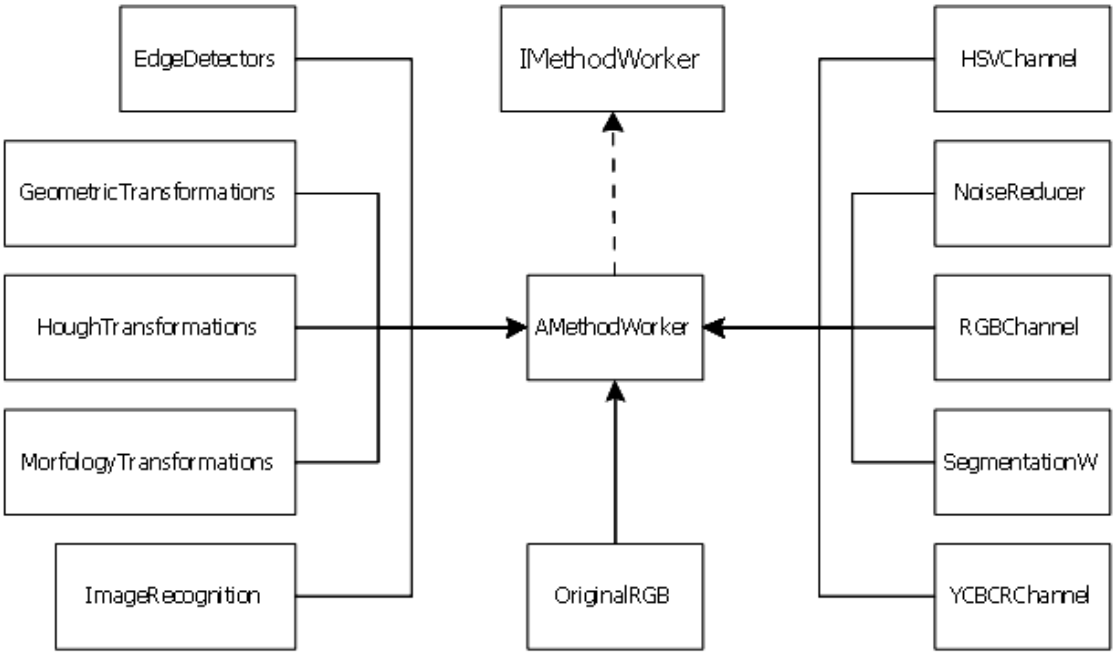
této metody jsou nastaveny cesty k obrázkům a je provedeno uložení do databáze. Implementací rozhraní IMethodWorker vzniklo v rámci této práce několik. Všechny spojuje společná abstraktní vrstva AMethodWorker, ve které jsou naimplementovány

```
public static IMethodWorker getMethod(String operationId) throws
IllegalInputException, ClassNotFoundException, IllegalAccessException,
InstantiationException {
  OperationRegister operationRegister= OperationRegister.getInstance();
  String className =operationRegister.getRelation(
                    operationId).getName();
  String classifier = operationRegister.getClassifier(operationId);
  IMethodWorker result =
        (IMethodWorker) Class.forName(className).newInstance();
  result.setClassifier(classifier);
  return result;
}
```

Zdrojový kód 6: Ukázka tovární metody třídy MehtodFactory

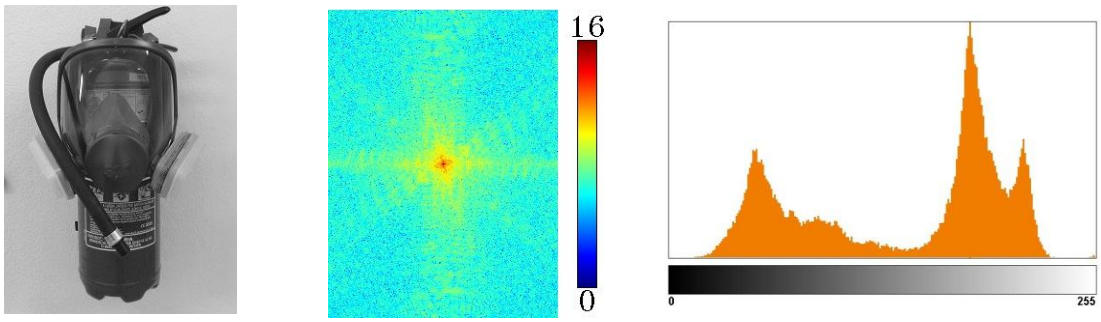
společné prvky.

Nejdůležitějším společným prvkem je vytvoření výsledných obrázků, tedy implementace metody saveImg. Tato metoda je zodpovědná za uložení dat po



Obrázek 27: Závislosti tříd na rozhraní IMethodWorker

dokončení zpracování aktuálně vykonané operace. Součástí toho je vytvoření histogramu a amplitudového spektra. Tato operace je provedena pro každou část sekvence.



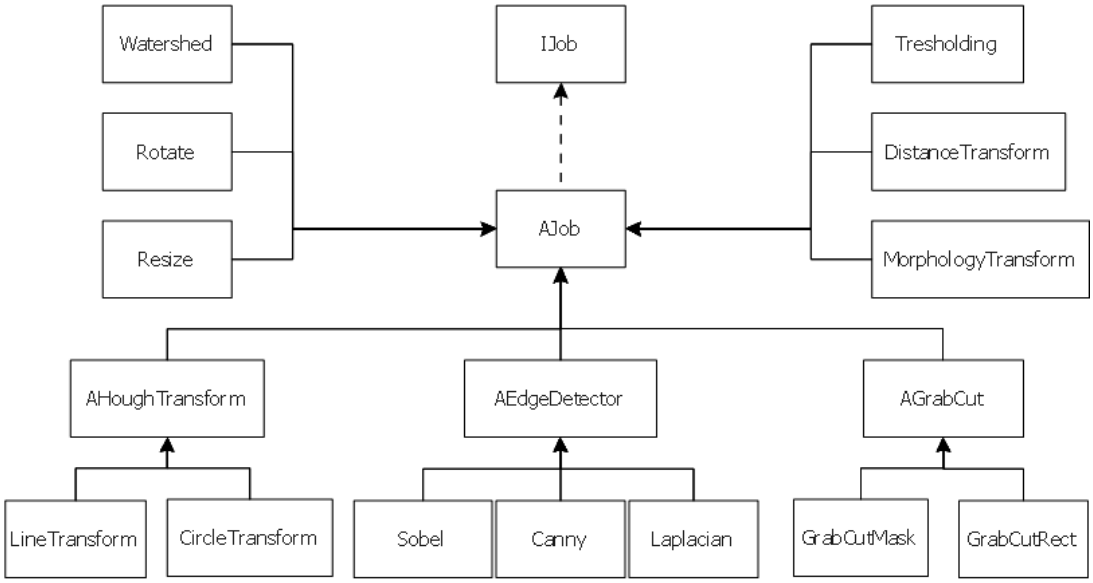
Obrázek 28: Výsledné obrazy po průchodu metodou saveImg. Z leva obraz převedený obraz z RGB do Gray jeho amplitudové spektrum a histogram

Nejdůležitější je vlastní implementace metody work. V této metodě je nadefinováno, co se s obrazem ve skutečnosti stane. Zde mohlo být zvoleno několik postupů, jak tuto implementaci provést. Prvním nejjednodušším způsobem by bylo vyhodnocení klasifikátoru a přímá implementace kódu v metodě work. Tento způsob je vhodný pouze pro jednoduché operace. Jeho výhodou je rychlý vývoj. Nevýhodou je znovu použitelnost kódu a s tím spojená jeho údržba. Tento způsob byl využit například při implementaci třídy zodpovědné za převod obrazu z RGB do jednotlivých vrstev.

```
public void work(){
    if (classifier.equals("očekávaný_klasifikátor") {
        Zda by byla přímo implementace.
}
```

Zdrojový kód 7: Ukázka přímé implementace kódu

Druhý způsob spočívá ve využití rozhraní IJob. Atribut toho typu je poskytnut z abstraktní vrstvy AMethodWorker a lze jej tedy využít v každé třídě, která je rozšířená touto abstraktní vrstvou. Implementace IJob potom umožňuje vyšší flexibilitu kódu, jelikož změna chování potom znamená využití jiné implementace tohoto rozhraní.



Obrázek 29: Závislosti tříd na rozhraní IJob

```
public void work() {
    if (classifier.equals("očekávaný_klasifikátor") {
        job = new Sobel();
        job.setPartAttributeValue(getAttributes());
        job.setImgData(imgData);
        setImgData(job.start());
    }
```

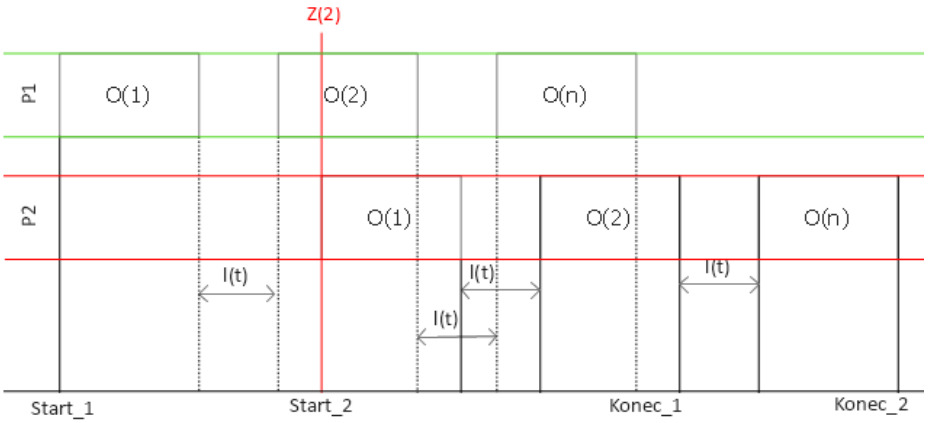
Zdrojový kód 7: Ukázka přímé implementace kódu

Třetí možností by potom mohlo být vytvoření tovární metody, která by pro jakýkoliv klasifikátor rozhodla jaká má být použita implementace rozhraní IJob. Tento postup by byl nejobecnější, avšak v rámci této práce zbytečný, a to z důvodu své robustnosti, a proto tato možnost nebyla implementována.

3.8.6 Aktualizace vykonávané sekvence

Při vytvoření a odeslání nové sekvence operací nelze počítat s tím, že by tuto činnost uživatel provedl vždy správně. Uživatel by tedy měl mít možnost v průběhu zpracování parametry sekvence upravovat, či dokonce do sekvence přidávat nové operace. Toto v zásadě lze řešit několika způsoby.

1) Nejjednodušším možným způsobem, který se naskýtá je pro každou změnu vždy vytvořit nový požadavek. S tímto řešením je však spojená vyšší zátěž na systém, a to z důvodu že pro každý požadavek je založena nová úloha, a to i za předpokladu, že původní úloha nebyla ukončena. Velikou výhodou tohoto přístupu je fakt, že není nutno řešit synchronizaci dat.

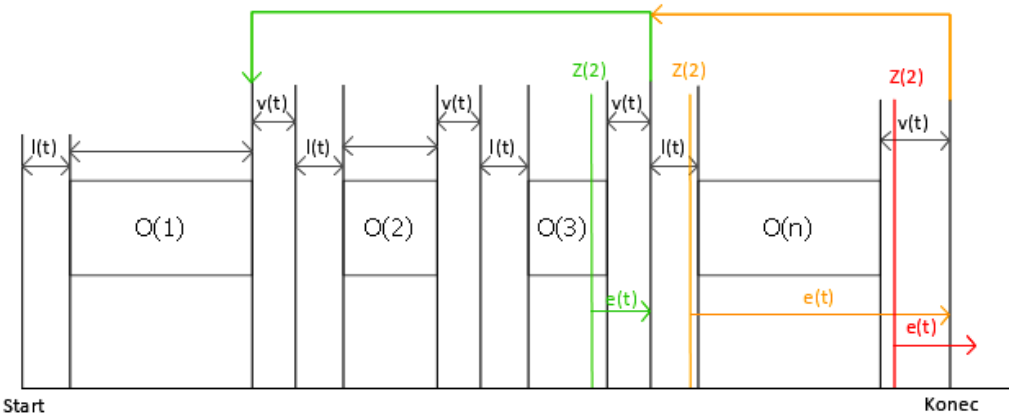


Obrázek 30:Průběh zpracování sekvence prvním způsobem

P1 označuje proces, který byl spuštěn jako první, P2 označuje proces, který byl spuštěn změnou parametru v druhé operaci prvního procesu, Z(2) označuje změnu druhé operace v prvním procesu. Na základě analýzy výše uvedeného

průběhu by bylo možné navrhnout zlepšení, které by spočívalo v ukončení vykonávání prvního procesu v případě změny. To bohužel v rámci tohoto systému není jednoduché, jelikož je procesu zpracování přiděleno náhodné vlákno, není tedy pak možné určit, které vlákno by mělo ukončit svoji činnost.

2) Druhou možností je zavést mezi jednotlivé vykonání operace synchronizační zónu. V této zóně by pokaždé došlo ke kontrole, zdali se vstupní data v průběhu zpracování neaktualizovala. V případě, že došlo k aktualizaci, vrátil by se proces k poslední vykonané operaci, která byla nezměněna. Zde tato myšlenka však naráží na několik problémů. Mějme například modelovou situaci, kdy sekvence obsahuje n operací. Menší problém tedy nastane, když je aktuálně zpracovávaná třetí operace a přijde aktualizace druhé operace. V tom případě část výpočtu proběhla naprosto zbytečně. Horší situace nastane, když přijde aktualizace v zóně, kdy probíhá načítání dat pro další operaci. Zde už není možné vykonání této operace zastavit. Změna by byla tedy detekována až po provedení celého výpočtu, což je o stupeň náročnější než předchozí případ. Nejzávažnější chyba by však mohla nastat v případě, že aktualizace příjde v průběhu, kdy dochází k ukládání a synchronizaci poslední operace. Poté tato aktualizace nemusí být aplikována a systém by mohl prohlásit, že aktualizace byla úspěšně aplikována i když nebyla.



Obrázek 31: Průběh zpracování sekvence druhým způsobem

Na výše uvedeném obrázku lze pak vidět schéma průběhu zpracování sekvence a její aktualizace. L(t) označuje dobu nutnou pro načtení dat, O(n) dobu vykonání n-té operace, v(t) označuje validační zónu, ve které dochází

k ukládání a synchronizaci dat, e(t) označuje dobu chybného výpočtu a Z(2) symbolizuje změnu parametru v druhé operaci.

V určitém pohledu lze považovat první variantu jako nejhorší scénář druhé varianty. Aktualizace první operace přijde v průběhu vykonávání poslední operace. Nutno ještě podotknout, že zpracování sekvence je prováděno asynchronně a bylo by tedy obtížně tuto informaci předat napřímo.

4 Testování

Jelikož se jedná o komplexní systém, musely být provedeny testy jednotlivých modulů i větších částí a na konec i celého systému v celku. Testování probíhalo na čtyřech zařízeních:

- telefon: Samsung S3 mini – OS android,
- telefon: Lumia 950XL – OS Windows 10,
- tablet:Samsung Galaxy Note 10.1 2014 OS Android,
- PC – OS -Windows.

Tato zařízení byla vybrána záměrně, aby každé disponovalo různým rozlišením a operačním systémem, a to z důvodu, že systém podporuje responzivní design. Bylo tedy nutné ověřit, že se vše zobrazuje na různých zařízeních korektně. Poté byl s každým zařízením proveden stejný test, který spočíval ve vytvoření sady stejných požadavků a odeslání na zpracování. Tento test byl prováděn proto, že velká část klientské aplikace byla napsána v JavaScriptu, který se občas na různých zařízeních chová různě.

4.1 Rest API

Pro testování rest API byla využita aplikace SOAP UI, která umožňuje odesílání požadavků a příjem jejich odpovědí. V rámci tohoto testování bylo sledováno, zdali se:

- spustí odpovídající operace při volání GET, POST, PUT, DELETE,
- v případě POST požadavku se data v odchozím objektu shodují s daty v deserializovaným java objektu,
- typ odpovědi (200, 404, 500) se typ odpovědi shoduje s očkovaným typem,
- se celková http odpověď shoduje s očekávanou odpovědí.

```
GET http://devtul.djin.cz:9080/rest/getFunctions HTTP/1.1
Accept-Encoding: gzip,deflate
Host: devtul.djin.cz:9080
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)
```

Zdrojový kód 8: Tělo testovacího http požadavku - získání všech funkcí

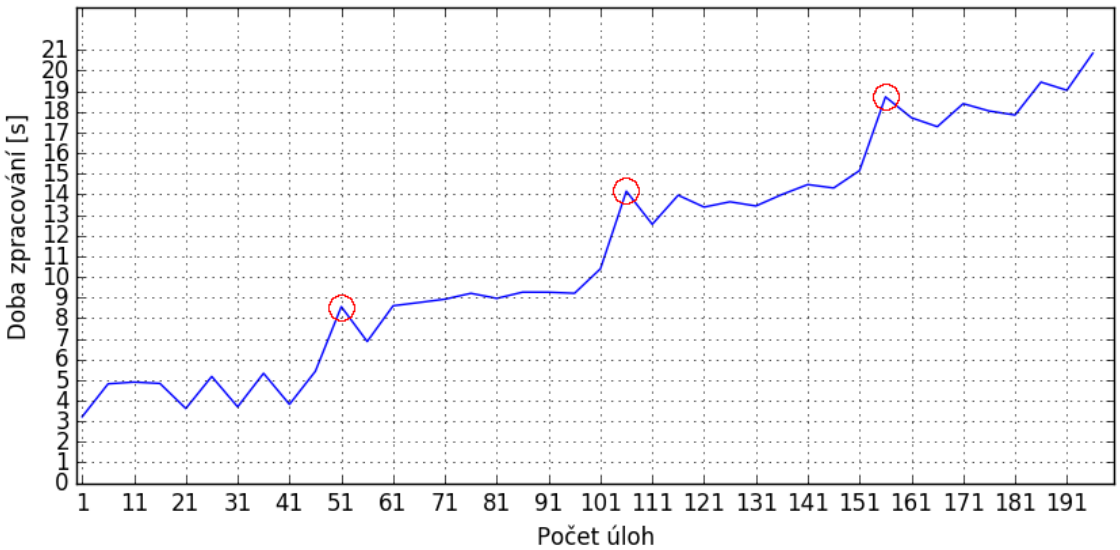
Ve výše uvedeném zdrojovém kódu 8 lze vidět hosta, na kterého bylo komunikováno, o jaký typ http požadavku byl použit a jaký byl použit agent.

```
    HTTP/1.1 200 OK
    Server: Apache-Coyote/1.1
    Content-Type: application/json;charset=UTF-8
    Transfer-Encoding: chunked
    Date: Wed, 03 May 2017 08:01:44 GMT
    [{"name":"Color Spaces","idMethod":"8c7b0461-f12c-482b-8139-
0874c48ffdcb"},{"name":"Edge Detectors","idMethod":"becfc423-5c98-
4fa8-889c-bb3b4eca092d"}, …]
```

Zdrojový kód 8: Tělo http odpovědi

4.2 Zátěžový test

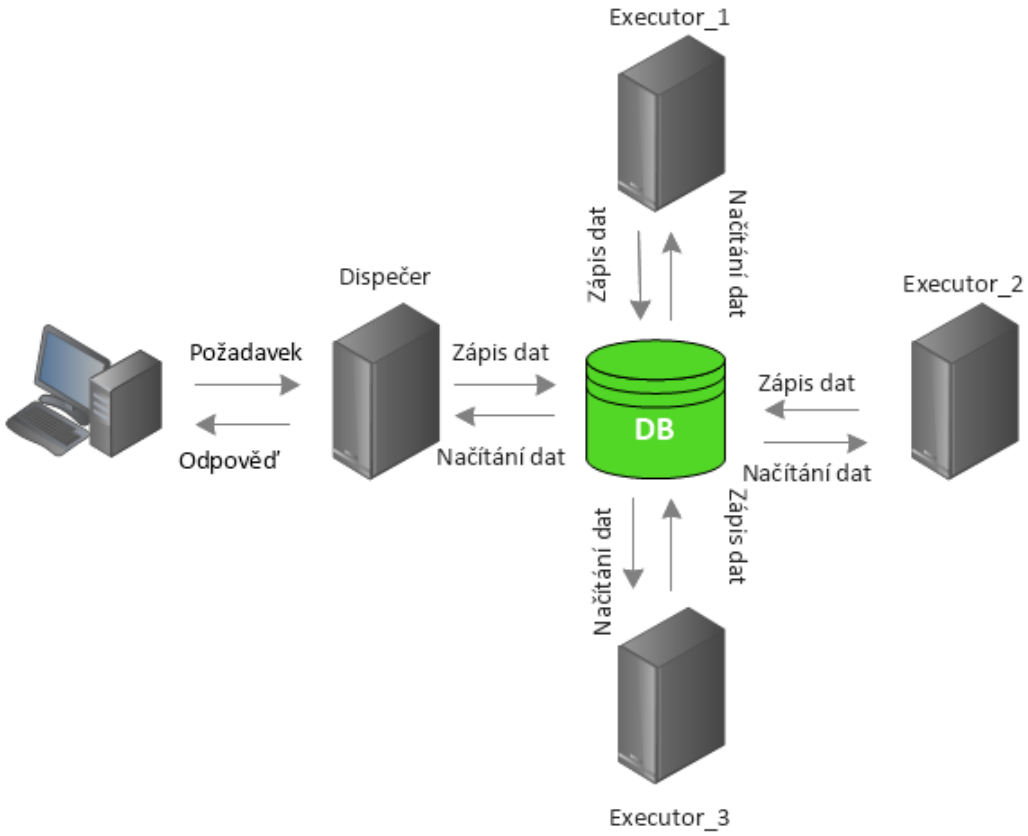
Aby bylo možné otestovat schopnost systému zpracovávat vyšší dávky požadavků na zpracování sekvence, byl v jazyce java vytvořen http klient, který generuje požadavky a měří čas, za který byla sada těchto požadavků zpracovaná. Aby byla měření objektivní, byla vždy využita stejná sekvence. Tato sekvence obsahovala čtyři operace - vytvoření základního obrazu v RGB, převod na šedotónový obraz, prahování a poslední operací, která byla na obraz aplikována, byla morfologická operace otevření.



Obrázek 32: Vyhodnocení testu

Na základě provedeného testu lze konstatovat, že systém splňuje základní požadavek, kterým je zpracování více požadavků zároveň. Z naměřených dat, která jsou uvedena na výše uvedeném obrázku, jsou znatelné tři velké nárůsty doby zpracování. Nárůst doby zpracování v těchto úsecích je dán založením požadavku do fronty a čekáním na jejich zpracování. Z toho výstupu je tedy znatelné, že systém je schopný

v současné konfiguraci odbavit zhruba 45 požadavků najednou, což je pro tento systém dostačující. Zvýšit počet zpracovaných sekvencí v jeden okamžik by bylo možné konfigurací systému. Druhou možností, jak zvýšit tento výkon, by byla distribuce mezi více serverů, na kterých by běželo toto zpracování. V tomto případě by pak musel být řešen problém se synchronizací tak, aby dva stroje nezpracovávaly stejnou úlohu. To by šlo vyřešit vytvořením dispečera, který by každé úloze přidělil identifikátor stroje, který jí smí zpracovat.



Obrázek 33: Alternativní návrh zlepšení výkonu

5 Možnosti využití

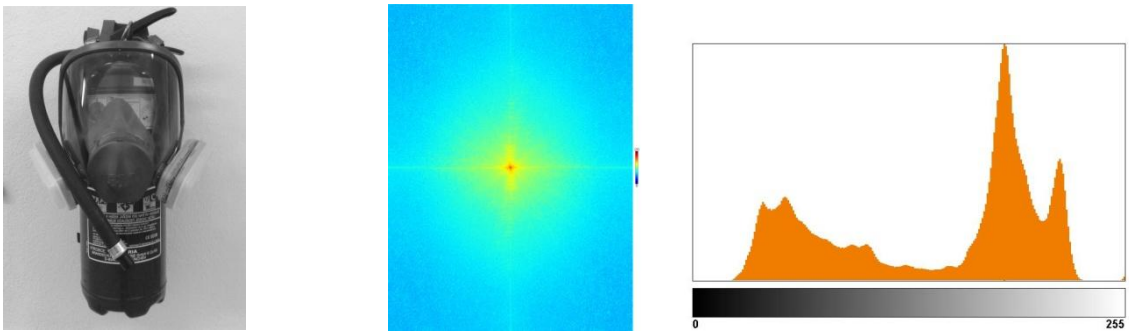
Tento systém by mohl být použit při výuce počítačového zpracování obrazu, kdy jsou studentům vysvětlovány základní operace s obrazem. Tento systém jim umožní si jednotlivé operace s obrazem prakticky vyzkoušet a porovnat výstup ze systému s výstupem ze svého programu.

5.1 Aplikace geometrických transformací

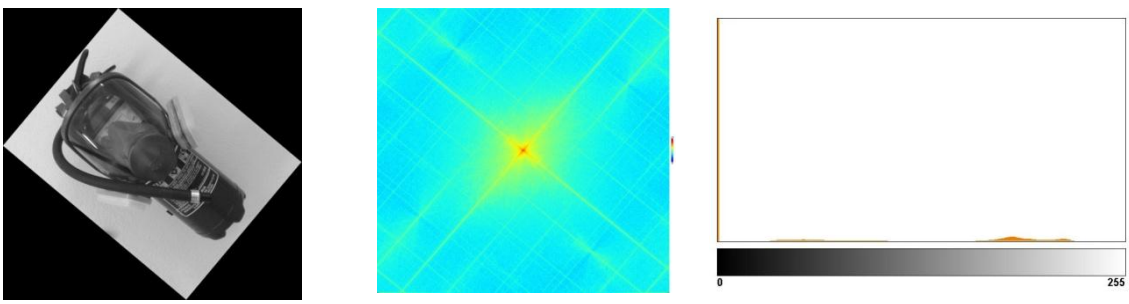
Základní geometrické transformace, které je možné s obrazem provést, jsou rotace zvětšení měřítka či zkosení.

Obrázek 35: Aplikace rotace na zelenou vrstvu obrázku

Na výše uvedeném obrázku lze vidět jak systém využít pro rotaci zelené vrstvy vstupního obrázku. Systém dále umožňuje u každého zpracovaného zobrazit histogram a amplitudové spektrum.



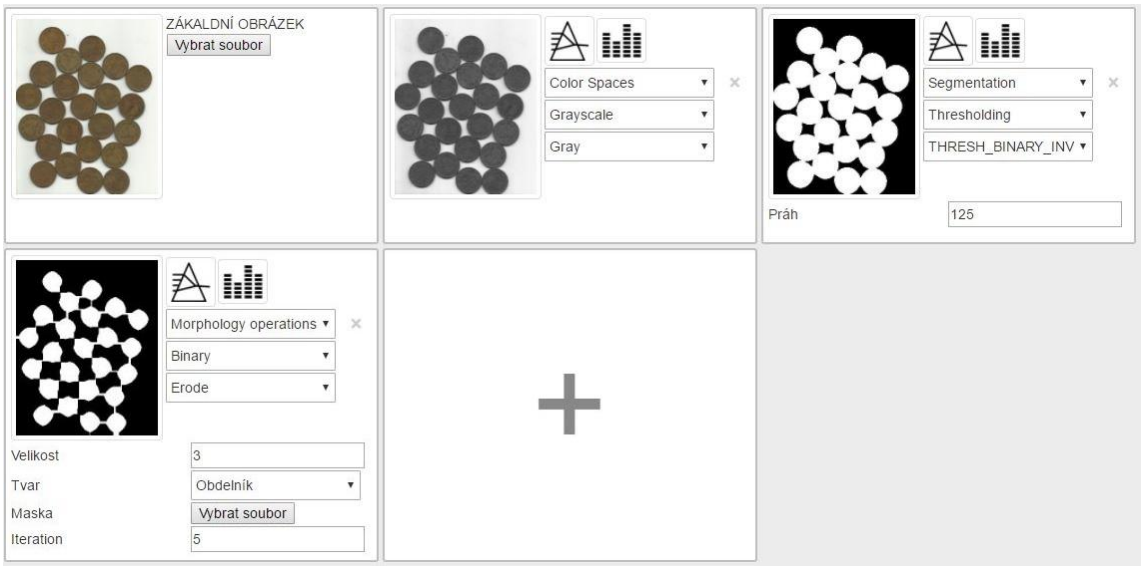
Obrázek 34: Zelená vrstva obrazu, jeho amplitudové spektrum a histogram



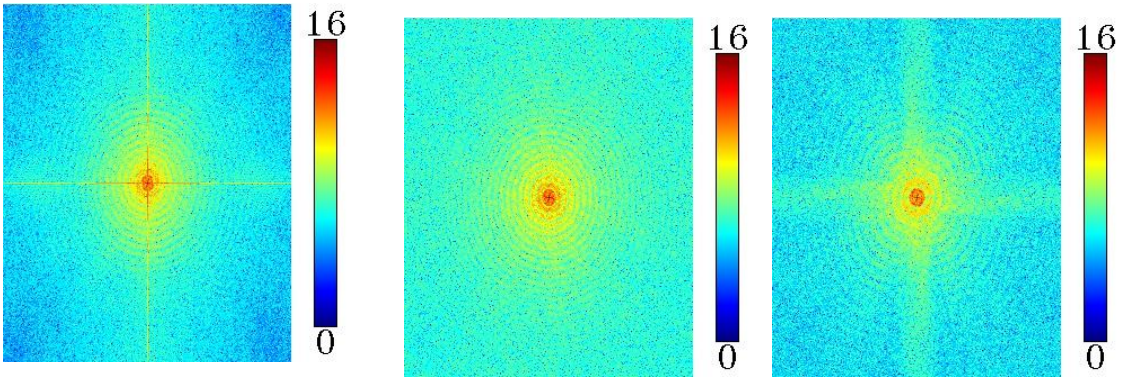
Obrázek 36: Rotace obrazu, jeho amplitudové spektrum a histogram

5.2 Aplikace morfologických operací

Tento systém též umožňuje praktické vyzkoušení morfologických operací jak s binárním obrazem, tak i šedotónovým. Morfologickým operacím lze též nastavit, kolikrát mají být na obraz aplikovány.



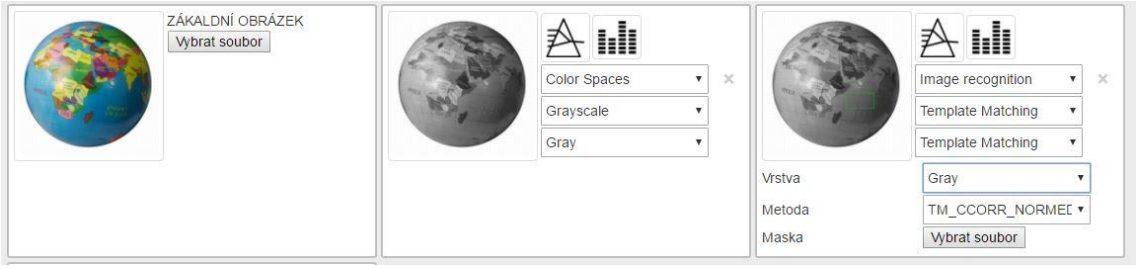
Obrázek 37: Ukázka zpracování morfologické operace eroze



Obrázek 38: Spektrum šedotónového obrazu, po provedení prahování a po provedení operace eroze

5.3 Aplikace metody Template Matching

Pomocí metody template matching lze vyhledávat přesně definované vzory. Vstupem do této metody je vyhledávaný vzor.



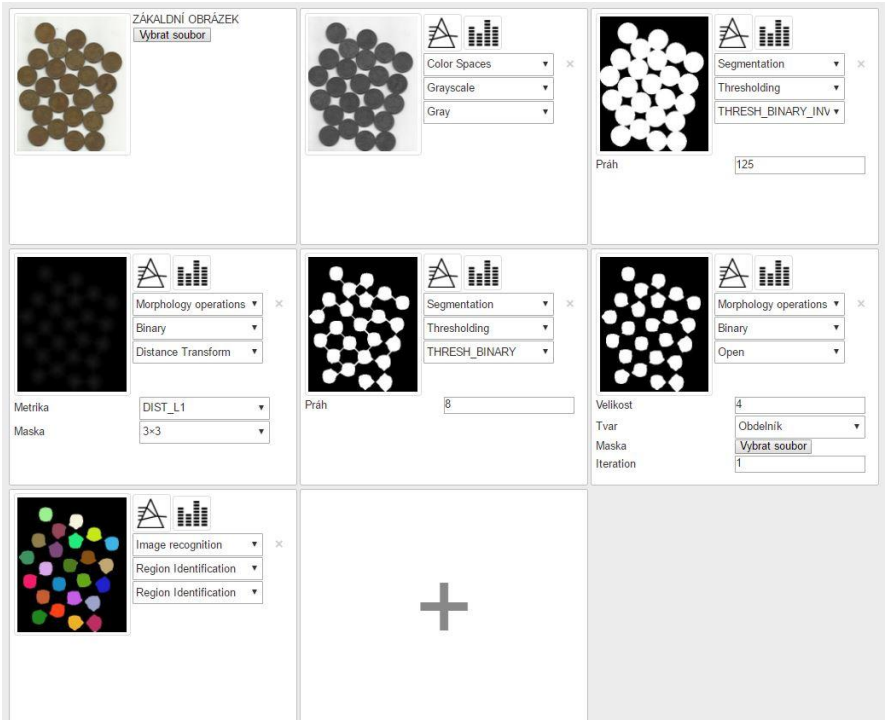
Obrázek 39: Ukázka užití metody Template Matching

Obrázek 40: Vstupní obraz, vyhledávaný vzor, nalezený vzor v obraze

5.4 Pokročilejší sekvence operací

U delších sekvencí je nutné brát v potaz, jak jsou nadefinované povolené kroky. Jak již bylo zmíněno, každá metoda nemůže být využita ve všech případech. Delší sekvence operací může být vytvořena například takto:

- převedení do gray,
- prahování s prahem T =125 (binary_inv),
- vzdálenostní funkce,
- prahování s prahem T = 8 (bingy),
- operace otevření s elementem velikosti 4,
- barvení oblastí.



Obrázek 41: Ukázka delší sekvence operací

Tohoto výsledku by šlo dosáhnout i snazší cestou, avšak zde šlo o demonstraci, že je možné operace řetězit tak, aby bylo dosaženo požadovaného výsledku.

6 Závěr

V rámci této diplomové práce byl navržen přívětivý webový systém pro poloautomatické zpracování obrazu. Tento systém umožňuje zpracování libovolně dlouhého lineárního řetězu operací. V rámci tohoto řetězu je prováděna validace posloupností jednotlivých kroků, jelikož se jedná o jeden ze stěžejních požadavků, který byl na tento systém kladen. Při návrhu systému byl také kladen důraz na rozšiřitelnost jeho funkčností. Z tohoto důvodu byly jednotlivé funkce navrženy jako moduly, které lze do systému snadno přidávat. Přidání nového modulu (funkce) tedy znamená zaregistrování modulu do registrů modulů. Na základě této registrace je potom systém schopný s novým modulem pracovat. Tomuto modelu musí být poté ještě nastaveno, které moduly (funkce) jej smí předcházet. Testy, kterým byl tento systém vystaven, bylo ověřeno, že je schopen v jednu chvíli nezávisle zpracovávat až 45 libovolných požadavků, což bylo vyhodnoceno jako dostačující. Další požadavky jsou poté zařazeny do fronty a jsou vykonány po dokončení některé z běžících úloh.

Při plnění cílů této diplomové práce byly získány cenné zkušenosti týkající se tvorby webových systémů v jazyce Java a knihoven, které byly v rámci vývoje využity. Největším problémem v rámci vývoje tohoto systému bylo správně zvolit metriku, podle které by mělo dojít k aktualizaci aktuálně vykonávaného řetězu operací. Ukázalo se, že nejefektivnější a nejjednodušší je vždy založit novou úlohu, a to i za předpokladu, že zpracování původní úlohy stále probíhá.

Další rozšíření této práce by mohlo spočívat v přidání administrativní sekce, ve které by bylo možné sledovat statistiky nejvyužívanějších metod, případně přidat možnost registrace uživatelů, na základě které by uživatel mohl sledovat svoji historii zadaných požadavků. Případně by se mohl systém rozšířit o údržbu, jejíž činností by bylo umazávání starých obrazových dat z úložiště a z databáze.

Použitá literatura

[1] DAVIES, E. R., 2005. *Machine vision: theory, algorithms, practicalities*. 3rd ed. Boston: Elsevier. ISBN 0122060938.

[2] DING, Lijun a Ardeshir GOSHTASBY, 2001. On the Canny edge detector. *Pattern Recognition*. **2001**(34), 721-725. ISSN 0031-3203.

[3] FENG, Liu, Liu XIAOYU a Chen YI, 2014. An efficient detection method for rare colored capsule based on RGB and HSV color space. In: *2014 IEEE International Conference on Granular Computing (GrC)* [online]. IEEE, s. 175-178 [cit. 2016-08-03]. DOI: 10.1109/GRC.2014.6982830. ISBN 9781479954643. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6982830

[4] HARALICK, Robert M. a Linda G. SHAPIRO, 1985. Image segmentation techniques. *Computer Vision, Graphics, and Image Processing* [online]. **29**(1), 100-132 [cit. 2016-08-03]. DOI: 10.1016/S0734-189X(85)90153-7. ISSN 0734189x. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S0734189X85901537

[5] HLAVÁČ, Václav. a Miloš. SEDLÁČEK, 2007. *Zpracování signálů a obrazů*. 2. přeprac. vyd. Praha: ČVUT. ISBN 9788001031100.

[6] JUN TANG, 2010. A color image segmentation algorithm based on region growing. In: *2010 2nd International Conference on Computer Engineering and Technology* [online]. IEEE, V6-634-V6-637 [cit. 2016-08-03]. DOI: 10.1109/ICCET.2010.5486012. ISBN 9781424463473. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5486012

[7] LAGANIÈRE, Robert., 2011. *OpenCV 2 computer vision application programming cookbook: over 50 recipes to master this library of programming functions for real-time computer vision*. 3rd. Birmingham, U.K.: Packt Open Source Pub. ISBN 9781786469717.

[8] LAGANIÈRE, Robert., 2011. *OpenCV 2 computer vision application programming cookbook: over 50 recipes to master this library of programming functions for real-time computer vision*. Birmingham, U.K.: Packt Open Source Pub.

[9] LELIS BAGGIO, Daniel, 2015. *OpenCV 3.0 Computer Vision with Java* [online]. Birmingham: Packt Publishing [cit. 2017-05-11]. ISBN 978-1-78328-398-9. Dostupné z: https://www.packtpub.com/application-development/opencv-30-computer-vision-java

[10] MAK, Gary., c2008. *Spring recipes: a problem-solution approach*. Berkeley, Calif.: Apress. ISBN 9781590599792.

[11] ŠONKA, Milan, Václav HLAVÁČ a Roger BOYLE, 2008. *Image processing, analysis, and machine vision*. 3rd ed. Toronto: Thomson. ISBN 9780495082521.

[12] VAN VLIET, Lucas J, Ian T YOUNG a Guus L BECKERS, 1989. A nonlinear laplace operator as edge detector in noisy images. *Computer Vision, Graphics, and Image Processing* [online]. **45**(2), 167-195 [cit. 2016-08-03]. DOI: 10.1016/0734-189X(89)90131-X. ISSN 0734189x. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/0734189X8990131X

[13] ZHANG, Hui, Quanyin ZHU a Xiang-feng GUAN, 2012. Probe into Image Segmentation Based on Sobel Operator and Maximum Entropy Algorithm. In: *2012 International Conference on Computer Science and Service System* [online]. IEEE, s. 238-241 [cit. 2016-08-03]. DOI: 10.1109/CSSS.2012.67. ISBN 9780769547190. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6394306

[14] OpenCv, 2016. *Open CV documentation* [online]. Itseez [cit. 2016-08-03]. Dostupné z: http://opencv.org/