VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

KONTROLA ZOBRAZENÍ TEXTU VE FORMULÁŘÍCH

QUALITY CHECK OF TEXT IN FORMS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. ZBYNĚK MORAVEC**
AUTHOR

VEDOUCÍ PRÁCE — **prof. Dr. Ing. PAVEL ZEMČÍK**
SUPERVISOR

BRNO 2017

Zadání diplomové práce/19700/2016/xmorav27

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání diplomové práce

Řešitel: **Moravec Zbyněk, Bc.**
Obor: Informační systémy
Téma: **Kontrola zobrazení textu ve formulářích**
Quality Check of Text in Forms
Kategorie: Zpracování obrazu

Pokyny:

1. Prostudujte literaturu na téma detekce geometrických tvarů a textu se zaměřením na 2D struktury formulářů zobrazovaných na displejích počítačů.
2. Navrhněte postup detekce textu a geometrických tvarů ve fotografiích formulářů snímaných kamerou nebo fotoaparátem z displeje počítače tak, aby bylo možné zjistit, zda texty jsou ve formuláři správně umístěny a zda "nepřetékají".
3. Navrhněte vhodný způsob implementace postupu a diskutujte jeho vlastnosti a možnosti. Současně navrhněte postup výběru nejvhodnějších fotografií tak, aby je bylo možno užívat v dokumentaci.
4. Implementujte postup a demonstrujte na vhodných příkladech funkčnost a vlastnosti implementovaného postupu.
5. Diskutujte dosažené výsledky a možnosti uplatnění a pokračování práce.

Literatura:

- Dle pokynů vedoucího

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3 zadání

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Zemčík Pavel, prof. Dr. Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 24. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Cílem této práce je kontrola správného zobrazení textů tlačítek na fotografiích displeje. Tyto fotografie obsahují různá zkreslení, která jsou komplikací pro následné rozpoznávání grafických elementů v obraze. Práce popisuje několik přístupů detekce formulářových tlačítek, dále popisuje implementovanou detekci, která využívá popisu tvarů kontur. Na takto nalezených tlačítkách je poté provedeno rozpoznání samotných vad zobrazení. Práce se také zaobírá automatickým výběrem nejkvalitnější fotografie k dokumentačním účelům.

Abstract

Purpose of this thesis is the quality check of correct button text display on photographed monitors. These photographs contain a variety of image distortions which complicates the following image graphic element recognition. This paper outlines several possibilities to detect buttons on forms and further elaborates on the implemented detection based on contour shapes description. After buttons are found, their defects are detected subsequently. Additionally, this thesis describes an automatic identification of picture with the highest quality for documentation purposes.

Klíčová slova

Houghova transformace, mean-shift, morfologické operace,konvoluční filtry, OCR, kontura, tvar, hodnocení kvality obrazu, OpenCV, C#, REST, Confluence

Keywords

Hough transform, mean-shift, morphological operators, convolution filters, OCR, contour, shape, image quality assessment, OpenCV, C#, REST, Confluence

Citace

MORAVEC, Zbyněk. *Kontrola zobrazení textu ve formulářích.* Brno, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Zemčík Pavel.

Kontrola zobrazení textu ve formulářích

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením prof. Pavla Zemčíka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Zbyněk Moravec
24. května 2017

Poděkování

Na tomto místě bych rád poděkoval vedoucímu mé diplomové práce prof. Pavlu Zemčíkovi za ochotné vedení této práce a ing. Jakubu Pavlákovi za konzultace ohledně požadavků zadání.

Obsah

Kapitola 1

Úvod

V dnešní době se lze setkat s dotykovými obrazovkami prakticky na každém rohu. Jsou používány nejen v osobních telefonech či tabletech, ale také v různých automatech či dokonce tiskárnách. Pomocí displejů může uživatel často snadno a intuitivně ovládat daná zařízení. S rostoucím počtem těchto zařízení, a také možností jimi poskytovaných, vzrůstají požadavky na automatizaci testování těchto uživatelských rozhraní. Tato práce se zabývá problematikou automatického rozpoznání špatně zobrazeného textu ve snímcích obrazovky, které byly pořízené fotoaparátem. Zvolené řešení je výhodné v tom, že je možné jej použít prakticky s jakýmkoliv vestavěným zařízením, které používá obrazovku pro interakci s uživatelem, a nevzniká tedy závislost na vnitřním fungování zařízení a jeho komunikaci s displayem. Na druhou stranu tyto fotografie obsahují jistá zkreslení (například rozmazání, odraz či moaré efekt) což znesnadňuje detekci požadovaných elementů. Druhým problémem, kterého se týká tato práce, je zvolení nejvhodnější fotografie pro dokumentaci. Výsledkem práce by tedy měla být aplikace, která omezí nutnost kontroly fotografií člověkem. Projekt vychází ze zadání společnosti `Y Soft Corporation`, která poskytla snímky obrazovek, které se nacházející v této práci ať už v originální či účelně upravené podobě.

Obsah této práce je rozdělen do dvou teoretických částí a jedné implementační. První část 2 se zabývá technikami týkajících se práce s obrazem o rozpoznání textu. Druhou kapitolu 3 tvoří popis nástrojů a knihoven použitých v práci. Třetí kapitola 4 se týká samotné implementace. Jsou zde popsány způsoby hledání textových tlačítek v obraze, včetně kontroly jejich zobrazení. Tato kapitola se také věnuje realizaci výběru a nahrávání fotografií do dokumentačního nástroje. Konec kapitoly je završen zhodnocením úspěšnosti detektoru zobrazení.

Kapitola 2

Práce s obrazem

V této části se budeme věnovat problematice práce s obrazem. Text níže slouží k popsání pojmů relevantních z hlediska této práce. Nesnaží se být encyklopedickým popisem všech možných postupů.

2.1 Digitální obraz

Jako obraz chápeme funkci dvou reálných proměnných `a(x, y)`, kde `a` reprezentuje amplitudu (například jas) obrazu na pozicích `(x, y)`. Digitální obraz `a[m,n]` popsaný ve 2D diskrétním prostoru je odvozen z analogového obrazu `a(x,y)`, který je reprezentován spojitým prostorem. Převod analogového obrazu na digitální je prováděn vzorkovacím procesem nazývaným také jako digitalizace. Dvoudimenzionální spojitý obraz `a(x,y)` je rozdělen do N řádků a M sloupců jednotlivých obrazových elementů, neboli pixelů (picture element). [19]

Používání digitálních obrazů je rozšířené, a proto bylo vytvořeno několik standardních formátů, které umožní sdílet data mezi různým softwarem a hardwarem. Níže je popis některých z nich. [22]

Formát Windows Bitmap Jedná se o jednoduchý formát podporující barevné obrazy s například indexovanými barvami, barvami v odstínech šedi či true-color. Volitelně tento formát podporuje jednoduchou bezztrátovou kompresi. [6]

Formát JPEG Tento formát umožňuje hardwarové kódování a dekódování v reálném čase. Podporuje obraz o velikosti až 64k x 64k pixelů a 24 bitové barevné hloubce. Tento formát umožňuje kompresi až 1:20 a to bez výrazné degradace. Komprese je účinná hlavně v případech, kdy obraz obsahuje velké oblasti s podobnou barvou. [22]

Formát GIF Tento formát byl hojně užíván pro kódování obrazů na webových stránkách. Nabízí však uložení pouze 256 barev. [22]

Formát PNG Cílem PNG bylo původně nahradit formát GIF kvůli jeho licenčním záležitostem. Tento formát byl navržen jako univerzální primárně pro užití na internetu. PNG podporuje několik typů obrazů, co se barev týče. Přidává však také možnost alpha kanálu, pro určení průhlednosti. Na rozdíl od 1 bitové průhlednosti ve formátu GIF nabízí PNG

16 bitů úrovní průhledností. PNG nabízí bezztrátovou kompresi, a tedy není konkurencí formátu JPEG. [6]

Formát TIFF Tag Image File Format je velmi obecný a komplexní formát, který je často používán obrazovými skenery. Podporuje obrázky s barevnou hloubkou 1 až 24 a umožňuje zvolit mezi ztrátovou či bezztrátovou kompresí. [22]

Technologie snímání obrazu

Mezi dva používané typy senzorů v digitálních fotoaparátech patří CCD a CMOS senzory. Na malém čipu jsou umístěny miliony fotosenzitivních diod, každá za účelem zachycení jednoho pixelu obrazu. Při sejmutí obrazu se závěrka fotoaparátu krátce otevře a každý pixel na senzoru obrazu zaznamená jas světla, které na něj dopadá. Poté, co se závěrka uzavře, se informace o intenzitě světla z každého pixelu zdigitalizuje. Tato tato informace je poté užita k rekonstrukci obrazu. [8].

CCD senzory

Zkratka CCD reprezentuje `charge-couplet-device`. Tato technologie byla vynalezena roku 1969 v Bellových laboratořích Willardem Boylem a George E. Smithem. CCD pro snímání obrazu se skládá z fotoaktivní oblasti a přenosové oblasti vyrobené z posuvného registru. Obraz je čočkami promítnut do fotoaktivní oblasti, kde je do kondenzátorového pole uložen elektrický náboj odpovídající dané intenzitě světla. Jakmile dojde k dostatečnému ozáření, řídící obvod způsobí, že obsah každého kondenzátoru je přesunut do sousedního kondenzátoru - jako v posuvném registru. Výstupní náboj posledního kondenzátoru je převeden na napětí. Postupně tedy získáme sekvenci napětí, která je pak obvykle vzorkována, digitalizována a následně uložena v paměti. [27]

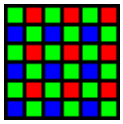
CMOS senzory

Největším problémem CCD senzorů je fakt, že je třeba užívat specializované a drahé procesy, které jsou použitelné pouze pro CCD, zatímco se vyrábí miliony čipů pro počítačové procesory a paměti technologií CMOS - Complementary Metal Oxide Semiconductor. Použití stejné technologie i pro výrobu obrazových senzorů může znatelně snížit náklady, neboť je možné cenu rozprostřít do většího množství zařízení. Další výhodou je, že obrazový senzor a zpracovávající obvod mohou být na stejném čipu. [8]

Mezi hlavní výhody CMOS senzorů patří nízká spotřeba energie, náhodný přístup k datům a například také rychlé snímkování. Na druhou stranu však CMOS trpí různými druhy šumů a získávají horší obraz než senzory CCD. [4]

Barevné snímání obrazu

Za účelem barevného snímání obrazu je obvykle používána Bayerova maska - 2.1. Ta se skládá ze 3 druhů filtrů, které propouštějící vždy jen červenou, modrou a zelenou. Filtrů propouštějící zelenou je dvakrát více než pro ostatní barvy. Výsledkem tohoto snímání je intenzita v každém pixelu. Výsledné rozlišení barevného obrazu je však nižší. [27]



Obrázek 2.1: Ukázka Bayerovy masky

2.2 Barva

Barva poskytuje informaci, která často zjednodušuje identifikaci objektu v obraze. Roku 1666 Sir Isaac Newton objevil, že když sluneční paprsek vstoupí do skleněného hranolu, vyšlý paprsek není bílý, ale skládá se ze spojitého spektra. To je tvořeno pozvolným přechodem mezi barvami fialové, modré, zelené, žluté, oranžové a červené.

Za barevné vidění oka mohou čípky. Ty lze rozdělit do tří hlavních kategorií podle specializace na červenou - 65%, zelenou - 33% a modrou - 2% (tyto čípky jsou nejcitlivější). Díky těmto charakteristikám lidského oka vnímáme barvu jako kombinaci tzv. primárních barev červené, zelené a modré (RGB). Smícháním primárních barev získáme sekundární barvy: purpurovou (červená+modrá), azurovou (zelenou+modrou) a žlutou (červená+zelená). [13]

Barevné modely

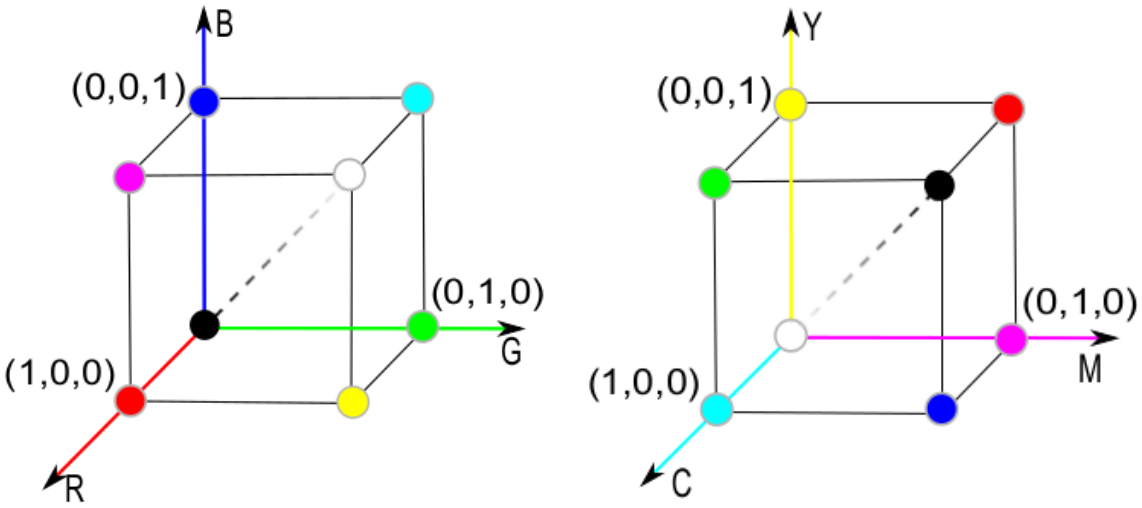
Barvu lze reprezentovat s užitím různých barevných modelů. Zde je krátký popis některých z nich.

Barevný model RGB Základními barvami tohoto modelu jsou barvy červená (R), zelená (G) a modrá (B). Ostatní barvy se vyjadřují aditivním skládáním barev - pomocí váhového součtu jednotlivých složek. Model RGB lze reprezentovat jednotkovou kostkou - 2.2a, kde jednotlivé osy reprezentují intenzitu základních barev v intervalu $\langle 0, 1\rangle$. Čím větší mají jednotlivé složky hodnotu, tím bude výsledná barva světlejší. Barvy na diagonále kostky mezi body $[0, 0, 0]$ a $[1, 1, 1]$ odpovídají barvám v odstínech šedi. Při implementaci je často místo spojitého intervalu $\langle 0, 1\rangle$ používáno diskrétních hodnot v rozsahu $\langle 0, 255\rangle$, které lze zakódovat pomocí 8 bitů. Tento model je používán barevnými monitory. [29]

Barevný model CMY Základními barvami jsou azurová (C), purpurová (M) a žlutá (Y). Tento model vychází z malířského míchání barev. Mícháním jednotlivých barevných složek získáváme výslednou barvu tmavších odstínů, jedná se tedy o substraktivní míchání. Míchání těchto barev si lze představit jako míchání jednotlivých filtrů, kde každá barva pohltí jinou část světla. Smícháním všech základních barev jsou pohlceny všechny barvy. Model CMY často reprezentujeme na jednotkové kostce - 2.2b, podobně jako v případě modelu RGB. Osy však reprezentují barvy C, M a Y. Transformace mezi modely RGB a CMY je poměrně přímá: [29]

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Barevný model CMYK Při tisku se výsledná barva nemíchá, ale barvy se postupně nanášejí v podobě vzorů natočených pod různým úhlem. Tento postup nezaručí dokonale



(a) Model RGB

(b) Model CMY

Obrázek 2.2: Znázornění barevných modelů na jednotkové kostce, převzato z [29].

černou barvu, ale například tmavě hnědou či tmavě šedou. Z tohoho a také ekonomického důvodu se proto používá další barva černá (K - key) - model CMYK. [29]

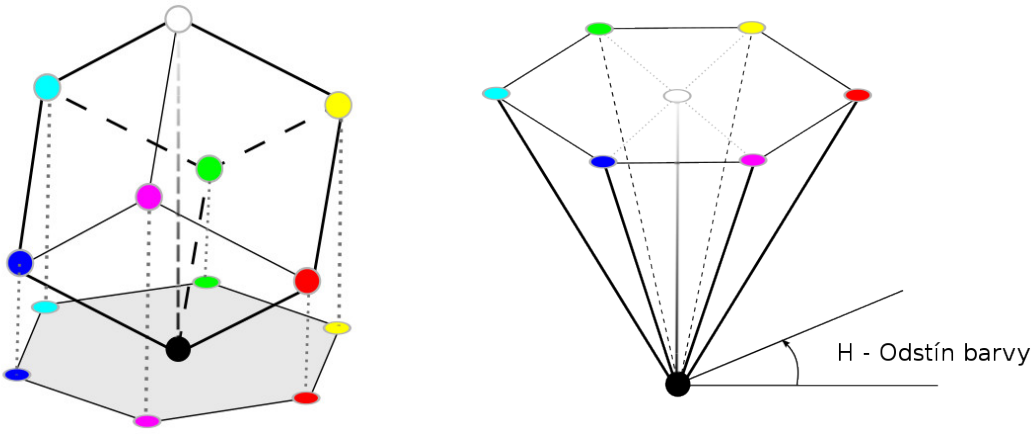
Uživatelsky orientované barevné modely Výše uvedené modely jsou orientované spíše na hardware. Na druhou stranu se používají také uživatelsky orientované modely skládající se z barevného odstínu (hue), sytosti barvy (saturation) a jasové složky (value, lightness, brightness, luminance). Tyto modely vycházejí z malířského způsobu barev a jsou často vhodné pro výběr barev v softwaru na editaci obrazu. Mezi představitele této skupiny patří například HSV, HSL a HSI. [29]

Například barevný model HSV používá pro definici barvy tři parametry. Prvním parametrem je odstín barvy (hue), který určuje dominantní spektrální barvu. Druhým parametrem je sytost (saturation), která určuje příměs ostatních barev. Třetím parametrem je jasová složka (value), která určuje množství bílého světla. Ke znázornění používáme pravidelný šestiboký jehlan, jak lze vidět na obrázku 2.3. [29]

2.3 Metody segmentace

Cílem segmentace je rozdělit vstupní obraz na jednotlivé regiony. Lze si ji představit jako proces, který rozděluje oblast celého obrazu `R` do `n` jednotlivých oblastí $R_1, R_2, ..., R_n$, tak aby platilo [13, s. 690]:

1. $\bigcup_{i=1}^{n} R_i = R$
2. R_i `je spojená množina bodů, i=1, 2, ..., n`
   - tedy všechny body v oblasti musí být určitým způsobem spojené (4/8-okolí)
3. $\forall i, j : i \neq j \Rightarrow R_i \bigcap R_j = \emptyset$
4. $Q(R_i) = TRUE, i = 1, 2, ...n$



(a) Projekce RGB kostky podél hlavní diagonály (b) Model HSV

Obrázek 2.3: Znázornění vztahu mezi modely RGB a HSV, převzato z [29].

- tedy pro všechny body musí být splněna nějaká vlastnost Q - například stejná intenzita.

5. $Q(R_i \bigcup R_j) = FALSE$, pro všechny přilehlé R_i a R_j

Segmentace může záviset na různých příznacích obrazu, jako je například barva či textura. Hlavním cílem segmentace je redukce informace pro snadnější analýzu. Metody segmentace mohou být založené na oblastech, hranách, prahování, shlukování na základě příznaků či na modelu. [21]

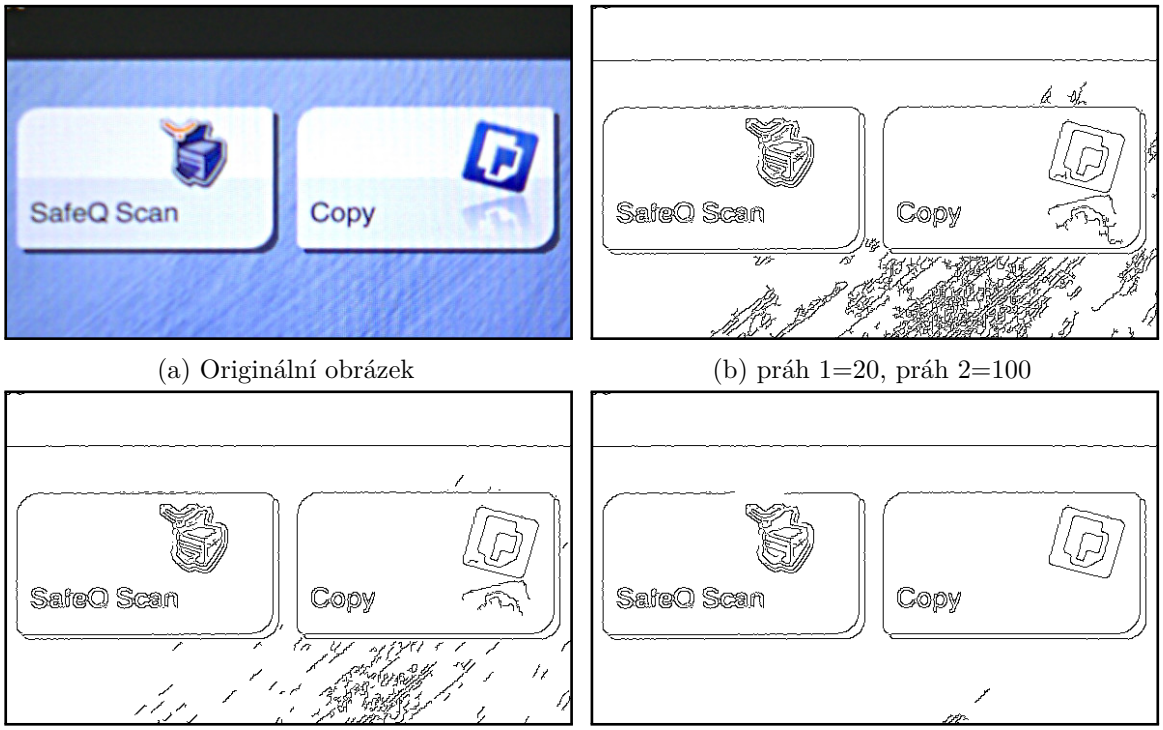
Canny detektor hran

Jedná se o velmi kvalitní detektor hran. Jeho cíle jsou [13, s. 719]:

- `Nízká chybovost` - Všechny hrany jsou nalezeny bez falešných hran.
- `Správná lokalizace bodu hran` - Vzdálenost mezi body hran a středy opravdových hran musí byt co nejmenší.
- `Detektor by měl vrátit pouze jeden bod pro každý bod hrany v obraze`

Jedná se o vícekrokový detektor. Jednotlivé kroky můžeme rozdělit do těchto fází [13, s. 723]:

1. Gausovská filtrace - tímto krokem se obraz vyhradí a zbaví se šumu.
2. Výpočet gradientu - získání velikosti a směru gradientu.
3. Potlačení nemaximálních částí gradientu
4. Použití dvojitého prahování a analýzy konektivity na detekci a spojení hran.



(a) Originální obrázek

(b) práh 1=20, práh 2=100

(c) práh 1=50, práh 2=100

(d) práh 1=50, práh 2=200

Obrázek 2.4: Aplikace Canny edge detektoru s různými parametry (Invertovaný výsledek)

Houghova transformace

Houghova transformace je technika, kterou lze využít k detekci přímek a křivek v obraze [22].

Pro popis Houghovy transformace a také pro účely této práce je nejvhodnější užití této metody pro hledání přímky. Analyticky lze přímku ve 2D prostoru definovat jako množinu bodů o souřadnicích x a y, pro které platí:

$$y = m * x + b \tag{2.1}$$

K Houghově transformaci je potřeba akumulátorové pole `A` o počtu dimenzí, které korespondují s počtem hledaných proměnných - v tomto případě 2. Nyní jsou procházeny všechny pixely obrazu. Pokud se jedná o pixel hrany, jsou získány vyhovující parametry přímek procházející tímto bodem - m, b. Ty jsou kvantovány do M, B. Když jsou takovéto hodnoty získány, je inkrementována hodnota akumulátoru A[M,B]. Po zpracování všech pixelů jsou v tomto poli hledána maxima, která by měla reprezentovat parametry přímek. [22]

Popis přímky pomocí $y = mx + b$ má však problém s popisem vertikálních přímek. Řešením této situace je reprezentace přímky pomocí následujícího vzorce [22]:

$$r = x * cos(\alpha) + y * sin(\alpha) \tag{2.2}$$

Kde `r` reprezentuje vzdálenost od počátku a α reprezentuje úhel mezi danou přímkou a x-ovou osou.

Z Houghovy transformace je odvozeno několik dalších metod. Jednou z nich je využití Houghovy transformace spolu s posuvným oknem [15] za účelem detekce obdélníků v obraze.

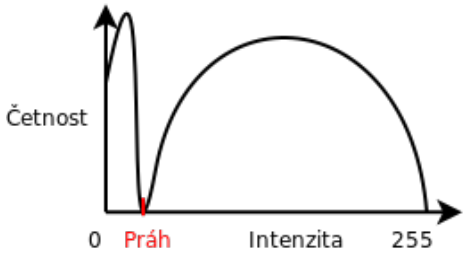
Základní idea tohoto algoritmu je užití posuvného okénka pro celý obraz. Toto okno by mělo být co nejmenší, ale zároveň dostatečně velké, aby se do něj vešel obdélník. Pro každou pozici takového okénka je vypočtena Houghova transformace za účelem nalezení obdélníku, který má střed totožný se středem posuvného okénka. V Houghově prostoru jsou poté hledána čtyři maxima mající mezi sebou určitý vztah. Tento vztah je určen na základě kolmosti hran obdélníku a na shodné vzdálenosti od středu (mezi dvěma páry hran).

Prahování

Prahováním je možné získat binární obraz z obrazu v odstínech šedi. Prahovací funkce poté rozhodne, které pixely tvoří popředí - oblast zájmu, a které jsou pouhým pozadím. Prahování může být založeno například na distribuci odstínů v obraze či na určité hodnotě. Jako oblast zájmu poté můžeme vybírat ty pixely, které jsou nad či pod daným prahem, nebo zda jejichž intenzita patří či nepatří do považovaného intervalu. [22]

Použití histogramu pro zvolení prahu

Volba prahu může záviset jak na manuálně zadané hodnotě uživatelem, tak být spočtena automaticky, například z histogramu obrazu. V nejjednodušším případě hledáme jediný práh, který odděluje světlé a tmavé pixely - jak lze vidět na umělém histogramu 2.5. [22]



Obrázek 2.5: Ukázka histogramu a automaticky zvoleného prahu

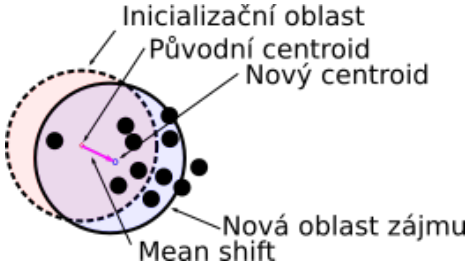
Mean-shift

Mean-shift je bezparametrická shlukovací metoda, která neklade požadavky na počet nebo tvar shluků [20]. Tento algoritmus má několik využití, v této práci se však budeme zajímat pouze o účel vyhlazení a segmentace obrazu [7].

Základní princip mean-shift lze rozdělit do čtyř kroků [16]:

1. Zvolení oblasti zájmu.
2. Určení centroidu ze vzorků v této oblasti. (mean)
3. Původní oblast zájmu se přesune na pozici tohoto centroidu. (shift)
4. Opakování bodu 2-4, dokud se nedosáhne konvergence.

Pro rozpoznání shluků tvořených jednotlivými vzorky, spustíme mean-shift pro každý vzorek v obraze. Získáme tak odpovídající lokální maxima hustoty pro tyto vzorky. Sjednotíme tato maxima, která se dle dané tolerance shodují. Shluky jsou nyní automaticky tvořeny vzorky, které by dokonvergovaly do společného maxima. Oblasti, z které vzorky dokonvergovaly do stejného bodu, říkáme "basin of attraction". [9]



Obrázek 2.6: Vizualizace kroku algoritmu Mean-shift

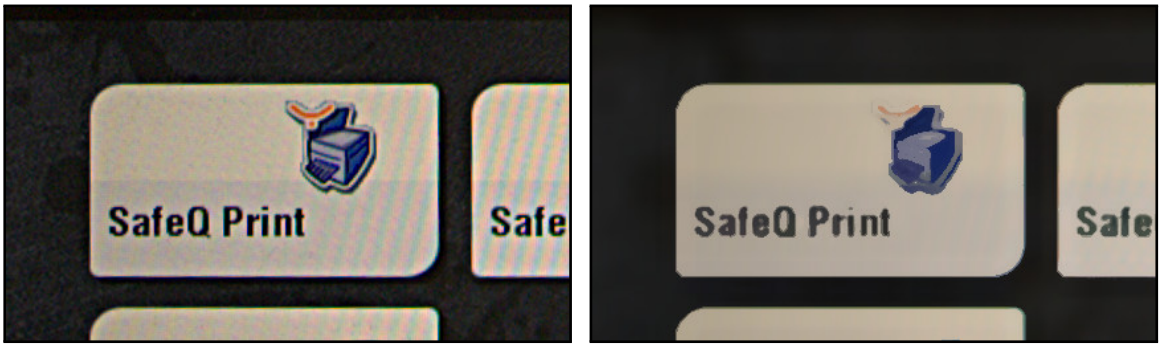
Filtrování a segmentace metodou Mean-shift

Jednou z možností, jak vyhladit obraz, je nahrazení hodnoty pixelu hodnotami (váženého) průměru pixelů v jeho okolí. To však kromě šumu odstraní také některé charakteristické informace. Vyhlazení zachovávající diskontinuitu však adaptivně redukuje úroveň vyhlazování blízko náhlých změn v lokální struktuře, tedy hran. [7]

Obraz si lze představit jako 2-dimenzionální mřížku p-dimenzionálních vektorů(pixelů). Například v případě šedotónového obrazu je p rovno 1, v případě barevného je p rovno 3. Prostor mřížky je chápán jako `spatial`[1] doména, zatímco úroveň intenzity či barva je chápána jako `range` doména. Pro oba tyto prostory je předpokládaná euklidovská metrika. Když se tyto domény spojí do spojité prostorově-barevné domény o d=p+2 dimenzích, je třeba provést normalizaci jejich rozdílných typů hodnot. [7]

Mean-shift filtrace pracuje nad pixely ve spojité spatial-range doméně. Pro každý vstupní pixel je vypočten bod kongruence pomocí metody mean-shift. Výsledná barva pixelu - range složka je určená složkou range bodu kongruence tohoto pixelu. [7]

Mean-shift segmentace je tedy pokračováním mean-shift filtrace, kdy jsou vytvořeny shluky filtrovaných pixelů dle jejich blízkosti v spatial a range doméně. [7]



(a) Obraz bez úpravy (b) Vyfiltrovaný obraz

Obrázek 2.7: Výsledek mean-shift filtrace

2.4 Morfologické operace

Mezi základní morfologické operace patří dilatace a eroze. [29, s.56]

[1] Původní anglický text používá spojení "spatial-range domain", pro které se nepodařilo najít vhodný překlad. Proto budou použity původní anglické výrazy.

Dilatace

Binární dilataci obrazu `F` a strukturálního elementu `S` definujeme takto [18, s. 70]:

$$C = F \oplus S = \{c | c = f + s, f \in F, s \in S\} \tag{2.3}$$

Efektem dilatace je rozšíření bílé oblasti. Touto operací je například možné odstranit některé nespojitosti v liniích, dojde ovšem také k zesílení tloušťky těchto linií.

Eroze

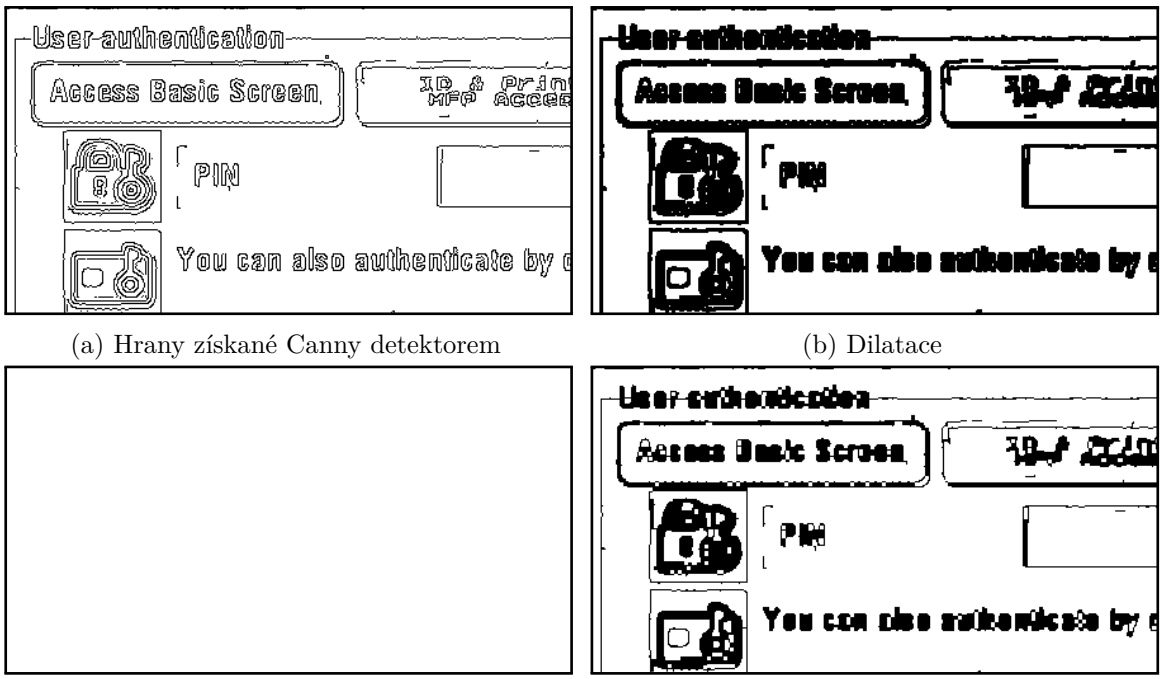
Jedná se o duální operaci k dilataci. Lze ji definovat takto [18]:

$$C = F \ominus S = \{c | (S)_c \subseteq F\} \tag{2.4}$$

Kde S_c znamená posunutí množiny S bodem c [18]:

$$(S)_x = \{r | r = s + x, s \in S\} \tag{2.5}$$

Eroze způsobí ztenčení bílé oblasti objektu. Kromě toho dojde k odstranění příliš malých objektů.



(a) Hrany získané Canny detektorem

(b) Dilatace

(c) Eroze

(d) Uzavření

Obrázek 2.8: Aplikace morfologických operací (Invertovaný výsledek)

Otevření a uzavření

Přestože efekty operací dilatace a eroze působí na první pohled jako inverzní, v praxi tomu tak obvykle není. Získáme-li například tenčí linii po operaci eroze, lze tuto linii opět rozšířit operací dilatace. Pokud však tato linie úplně zanikne, operace dilatace již nemá možnost

s touto linií nadále pracovat. V případě, že nejdříve aplikujeme erozi a poté dilataci s využitím stejného strukturálního objektu, tedy

$$((F \ominus S) \oplus S) \tag{2.6}$$

získáme operaci zvanou morfologické otevření [29]. Provedeme-li posloupnost operací v opačném pořadí, tedy nejdříve dilataci a poté erozi,

$$((F \oplus S) \ominus S) \tag{2.7}$$

získáme operaci zvanou morfologické uzavření [29].

Výsledky morfologických operací lze vidět na obrázcích 2.8. Na obrázku 2.8c dochází vlivem eroze k odstranění všech hran v obraze.

2.5 Konvoluční filtry

Konvoluce je důležitá operace nejen v oblasti zpracování obrazu. Matematicky lze diskrétní konvoluci dvou proměnných zapsat jako [29]:

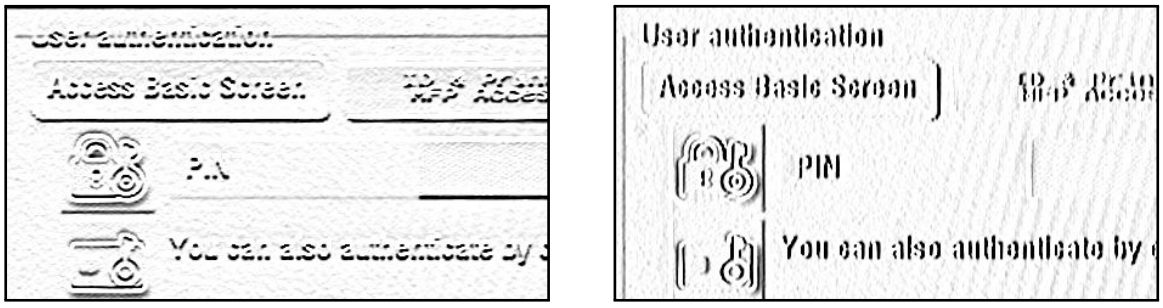
$$g(x,y) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m,n)h(x-m, y-n) \tag{2.8}$$

`f(x,y)` reprezentuje vstupní obraz a `h(x,y)` reprezentuje filtrační jádro nazývané také jako filtrační či konvoluční maska. Je však možné použít i zkrácený zápis[29]:

$$f(x,y) \star h(x,y) \tag{2.9}$$

Výpočet konvoluce si lze představit jako pokládání filtru na všechny vstupní pixely obrazu a vynásobení odpovídajících pixelů vstupního obrazu a filtru, následně uložení sumy těchto násobení do odpovídajícího výsledného pixelu. Dle volby vstupního filtru jsme schopni nad obrazem provést různé operace, například rozostření, zvýraznění hran apod. Jedním z používaných hranových filtrů je Sobelův filtr, níže lze vidět jeho vertikální formu. [29]

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -2 \end{bmatrix}$$



(a) Zvýraznění horizontálních hran

(b) Zvýraznění vertikálních hran

Obrázek 2.9: Aplikace Sobelova operátoru na fotografii tlačítek (invertovaný výsledek)

Výsledky aplikace Sobelova filtru lze vidět na obrázku 2.9, který znázorňuje jak jeho horizontální, tak vertikální variantu.

2.6 Rozpoznávání textu v obraze

Historii vývoje systémů pro rozpoznávání znaků lze rozdělit do generací [11]:

Komerční OCR systémy, které se objevily mezi lety 1960 a 1965 nazýváme první generací OCR. Tato generace je charakterizována schopností rozpoznání znaků s omezenými tvary, které mohly být čteny. Symboly byly speciálně navrženy pro strojové čtení a první z nich nevypadaly vůbec přirozeně. [11]

Druhá generace čtecích zařízení se objevovala mezi léty 1965 až do raných sedmdesátých let dvacátého století. V této době byl společností Toshiba vyvinut první automatický třídící stroj, který třídil dopisy podle poštovního čísla. Byl definován americký standardizovaný typ písma OCR-A a evropský standard OCR-B. Tyto fonty byly navrženy tak, aby byly snadno čitelné jak strojem, tak člověkem. [11]

Třetí generace OCR systémů se objevovala během sedmdesátých let dvacátého století. Cílem rozpoznávání byly například dokumenty ve špatné kvalitě a ručně psané znaky. Dalšími požadavky byla také nízká cena a vysoký výkon. Přestože se na trhu začaly objevovat více sofistikované OCR systémy, jednoduchá zařízení byla stále užitečná - umožňovala načítat texty psané na psacím stroji a následně jejich finální úpravu v počítači. [11]

Dle původního rozdělení do čtyř generací podle [11], je čtvrtá generace OCR považována za „dnešní“. Autor [2] v roce 2009 popisuje poslední čtvrtou generaci například těmito vlastnostmi: Rozpoznávání znaků v komplexních dokumentech obsahující text, grafiku, tabulky, matematické symboly, ručně psané znaky, dokumenty nízké kvality atd. Na velkém množství článků a patentů je úspěšnost až 99,99%.

Prakticky je možné se dnes setkat s komplexními systémy na rozpoznávání textů pro čtení tisknutých dokumentů spolu s převodem do editovatelných formátů kancelářských balíků. Z takto zpracovaných dokumentů je rozpoznáván kontext textu, například zda-li se jedná o text v patičce, či v tabulce. [1]

Postup rozpoznání textu

Činnost některých OCR řešení může být popsána následujícími fázemi [14]:

První fází je převod získaného obrazu do digitální podoby. Takový obraz poté bývá předzpracován. Předzpracování obvykle převede obraz do binární podoby - černá a bílá. Za účelem zvýšení kvality následného rozpoznání textu bývají také provedeny operace a to například filtrace šumu a vyplnění děr. [14]

Získaný a předzpracovaný obraz je poté rozsegmentován do několika symbolů. Obvykle tato segmentace probíhá hierarchicky. V první řadě jsou rozsegmentovány řádky s využitím histogramu řádků. V druhé řadě jsou rozsegmentována jednotlivá slova a nakonec písmena. Správně provedená segmentace má velký vliv na celkový výsledek. [14]

Z rozsegmentovaných symbolů jsou poté získány jednotlivé příznaky, které jsou poté využity pro trénování systému. Mezi tyto techniky patří například histogram projekce. Histogram horizontální projekce zobrazuje, kolik pixelů rozpoznávaného znaku existuje na daném řádku a vice versa. Pokud budeme používat histogramy s konstantním rozlišením, dosáhneme invariance vůči velikosti znaků. Následné porovnání znaků je tedy založeno na rozdílech získaných histogramů. [14][10]

Z rozpoznaných příznaků se poté OCR systém snaží zařadit znak do správné třídy. Moderní OCR systémy jsou často založeny na matematickém formalizmu, jenž se snaží minimalizovat míru chyby klasifikace. Mezi tyto přístupy patří Bayesův klasifikátor, diskriminační analýza či umělé neuronové sítě. [5][14]

Poslední stádium bývá označované jako post-processing. Cílem této fáze je detekovat a opravit jazykově nesprávné konstrukce. Obecně se jedná o chyby ze dvou kategorií. V prvním případě rozpoznané slovo neexistuje ve slovníku slov daného jazyka. V druhém případě se slovo ve slovníku sice nachází, ale v daném kontextu nemá smysl. [3]

2.7 Zhodnocení kvality obrazu

Zhodnocení kvality obrazu lze rozdělit do tří kategorií: NR (No reference - bez přístupu ke vzoru), FR (s přístupem ke vzoru) a RR (reduced reference). V případě hodnocení kvality s RR máme k dispozici kromě „nekvalitního“ obrazu také příznaky vzoru. Poté je možné na základě porovnání extrahovaných ze vzoru a rysů získaných z obrazu zhodnotit kvalitu obrazu. [24]

V mnoha aplikacích pro hodnocení kvality obrazu není přístup k referenčním vzorům. Přestože člověk umí v této situaci zhodnotit kvalitu obrazu, ukazuje se, že hodnocení kvality je v tomto případě velmi obtížným úkolem. Přístupů k bez-referenčnímu hodnocení kvality je mnoho. Většina z nich se snaží hodnotit obraz z hlediska ostrosti, šumu, a zkreslení vzniklého kompresí (blokování a prstencování). [28]

Bez-referenční percepční metrika pro neostrost

Jedno z potenciálních řešení hodnocení je prezentováno autory článku [17].

Měření neostrosti je definováno v prostorové doméně. Neostrosti lze vidět na hranách a oblastech textur. Prezentovaná technika je založena na měření šířky hran, kde platí: čím je hrana širší, tím víc lze považovat obraz za rozmazaný. Přestože mají hrany v obraze různou orientaci, autoři zpozorovali, že pro praktické užití stačí sledovat pouze hrany vertikální. Algoritmus je následující:

- aplikace hranového detektoru (například Sobelův filtr)
- procházení jednotlivých řádků obrazu
- pro každou hranu se určí její šířka, která je považována za lokální rozostření
- určení globálního rozostření na základě průměrů lokálních rozostření

Autoři také slibují vysokou korelaci mezi výsledky metody a subjektivním hodnocením.

Kapitola 3

Vývojové a další IT technologie

Tato kapitola obsahuje stručný přehled relevantních technologií z hlediska tohoto projektu.

3.1 Programovací jazyk C#

Jazyk C# je multiparadigmový jazyk pro obecné použití vyvinutý společnosti Microsoft. V roce 1990 byl vytvořen tým lidí za účelem vytvoření nového jazyka, který se jmenoval Cool (C-like Object Oriented Language). Později však Microsoft od tohoto názvu upustil a jazyk přejmenoval na C#. Toto jméno je inspirováno hudební notací, kde křížek indikuje zvýšení o půltón. Jazyk je silně typovaný s podporou implicitního typování pomocí užití klíčového slova `var`. Na rozdíl od jazyka Java je třeba v případě přepisování (angl. overriding) metod během dědění tříd využit klíčového slova `virtual` u přepisované metody, jako například v jazyce C++. [26]

V jazyce C# je možné použití ukazatelů pro přístup do paměti. To pouze v případě, že daný blok je označen jako `unsafe`. To také vyžaduje, aby program s tímto blokem měl příslušné oprávnění pro start. Kód, který není označen jako `unsafe`, stále může s ukazateli v podobě typu `System.IntPtr` manipulovat, nemůže je však dereferencovat. Většina přístupů k objektům obvykle ale probíhá pomocí bezpečných referencí, kde je vždy zaručena buď validní reference, a nebo definovaná `null` hodnota. [26]

Mezi používané integrované vývojové prostředí patří `Microsoft Visual Studio` pro operační systém Windows. Pro uživatele Linuxu je pak k dispozici `MonoDevelop` či nově vznikající multiplatformní vývojové prostředí `JetBrains Rider`.

Jazyk C# nabízí mnoho zajímavých konstrukcí, které sice nezvyšují jeho schopnosti, často ale zjednodušují práci programátora. Syntaktickým cukrem z hlediska rozšiřování objektů o další funkce jsou rozšiřující metody (angl. extension methods). To umožňuje programátorovi definovat statické metody třídy, které poté programátor může použít jako metody objektu.

Kód 3.1: Ukázka extension metod.

```
static class ExtensionClass {
  public static int Double(this int val){
    return 2 * val;
  }
}

class MainClass
```

```
{
  public static void Main (string[] args)
  {
    // vlastni zavolani metody
    int val2 = 1.Double ();
  }
}
```

Dalším způsobem pro zjednodušení práce je používání vlastností objektu (angl. properties), které umožňují vyhnout se velkému množství kódu nutného pro psaní přístupových metod pro práci s atributy objektu.

Kód 3.2: Ukázka properties.

```
class Objekt{
  public int Vlastnost {get; protected set;} // "getter" a
     "setter"
  private int _hodnota;
  public int Vlastnost2 {
    get { return _hodnota; } // "getter" atributu _hodnota
  }
}
```

3.2 Knihovna OpenCV

OpenCV[1] (Open Source Computer Vision Library) je otevřená knihovna pod BSD licencí pro počítačové vidění a strojové učení. Součástí knihovny je více než 2500 optimalizovaných algoritmů. Tyto algoritmy mohou být použity například k sledování pohyblivých objektů, rozpoznání obličejů, identifikaci objektů. Komunita knihovny má více než 47 tisíc lidí. Pomoc s jejím použitím poskytne také dokumentace s množstvím příkladů a ukázek kódů.

Samotná knihovna je napsaná v jazyce C++ a lze ji oficiálně použít v jazycích C, C++, Java, Python. Existují ale i knihovny třetích stran, které umožňují práci OpenCV i například v jazyce C#. Mezi knihovny třetích stran patří například Emgu CV či OpenCvSharp, která byla použita v této práci. Oficiální ukázka[2] knihovny OpenCvSharp:

Kód 3.3: Použití knihovny OpenCvSharp.

```
using OpenCvSharp;

class Program
{
  static void Main()
  {

    // Nacteni obrazu
    Mat src = new Mat("lenna.png", ImreadModes.GrayScale);
    Mat dst = new Mat();

    // Aplikace Canny detektoru
    Cv2.Canny(src, dst, 50, 200);
```

[1] http://opencv.org/about.html
[2] https://github.com/shimat/opencvsharp

```
        // Zobrazeni
        using (new Window("src image", src))
        using (new Window("dst image", dst))
        {
          Cv2.WaitKey();
        }
      }
    }
```

3.3 Dostupná řešení pro detekci textu

Rozpoznáváním textů se zabývá mnoho aplikací. Často se jedná o nástroje umožňující detekci textů v naskenovaných dokumentech. Tyto, často komerční řešení bohužel neposkytují API, které by bylo možné využít. Existuje[3] však několik knihoven poskytujících rozhraní pro různé jazyky. Dále budou zmíněny některé s neproprietárních řešení s možností použití v jazyce C# či alespoň C/C++.

Tabulka 3.1: Seznam svobodných OCR knihoven

	Tesseract	**Puma.NET**	**CuneiForm**	**GOCR**	**Ocrad**
Licence	Apache	BSD	Free	GPL	GPL
Poslední verze	16.2.2016	29.10.2009	6.6.2013	5.3.2013	8.4.2015

CuneiForm[4] Podle [25] se jedná o původně komerční produkt ruské společnosti Cognitive Technologies, jehož kód byl v roce 2008 otevřen. Oficiální web je bohužel od roku 2014 nefunkční[5] a nelze získat podrobnější ověřené informace.

Puma.NET[6] Puma.NET používá výše zmíněný CuneiForm jako své jádro. Cílem této knihovny je umožnit jednoduché začlenění OCR funkcionality do aplikací využívajících .NET framework. Je slibována podpora téměř všech tisknutých stylů písma s podporou detekce formátování - velikost, kurzíva, tučnost a další. Knihovna je schopna rozpoznat text v 27 různých jazycích spolu s kontrolou pravopisu a automatické korekce.

Ukázka použití [7]:

Kód 3.4: Ukázka jednoduchosti použití Puma.NET.

```
var pumaPage = new PumaPage("page001.jpg");
using(pumaPage)
{
  pumaPage.FileFormat = PumaFileFormat.RtfAnsi;
  pumaPage.EnableSpeller = false;
  pumaPage.Language = PumaLanguage.English;
}
pumaPage.RecognizeToFile("page001.rtf");
```

Poslední verze knihovny byla vydána v roce 2009 ve verzi beta.

[3] https://tinyurl.com/ocr-comparison
[4] https://en.wikipedia.org/w/index.php?title=CuneiForm_(software)&oldid=754913807
[5] Poslední funkční verze v archivu https://tinyurl.com/archive-org-cunei
[6] http://pumanet.codeplex.com/
[7] http://www.codeplex.com/Download?ProjectName=pumanet&DownloadId=89741

GOCR[8] Přestože je originální jméno projektu GOCR, je také nazýván JOCR z historických důvodů obsazenosti domény v době její registrace. Jedná se o program pod GNU GPL licencí, který nabízí možnosti čtení mnoha formátů obrazových souborů, jako např. pnm, pbm, pgm, ppm, pcx a tga. Kromě užití aplikace za samotným rozpoznáním znaků je možné použití k rozeznání čárových kódů. Aplikace nabízí grafické rozhraní napsané v tcl/tk. Programové rozhraní je poskytováno knihovnou libgocr, která je ovšem od roku 2006 považována za mrtvou[9] a měla být nahrazena projektem **Conjecture**.

Ocrad[10] Nástroj GNU Ocrad je založen na metodě extrakce příznaků. Jednou z jeho částí je také analyzátor rozvržení, který je schopen rozdělit nalezený text do sloupců či bloků. Projekt nabízí rozhraní jak ve formě konzolové `ocrad` aplikace, tak ve formě knihovny.

Tesseract[11] Oficiální projekt se skládá ze dvou částí: `libtesseract` a konzolové aplikace `tesseract` . Tesseract byl původně vyvíjen v laboratořích Hewlett-Packard, později byl jeho kód otevřen. Nyní je vyvíjen firmou Google. Tesseract poskytuje rozhraní pro velké množství programovacích jazyků, jako například C, C++, Java, Python, C#. Součástí nástroje jsou specifická data pro každý jazyk, ta však lze rozšířit o vlastní. V současné době se používá třetí verze této knihovny. Je však k dispozici i čtvrtá verze ve stádiu alpha, která využívá LSTM neuronové sítě. Knihovna umožňuje získávání textu z obrazu po různě velkých celcích, jako například po blocích, odstavcích, řádcích, slovech, či symbolech.

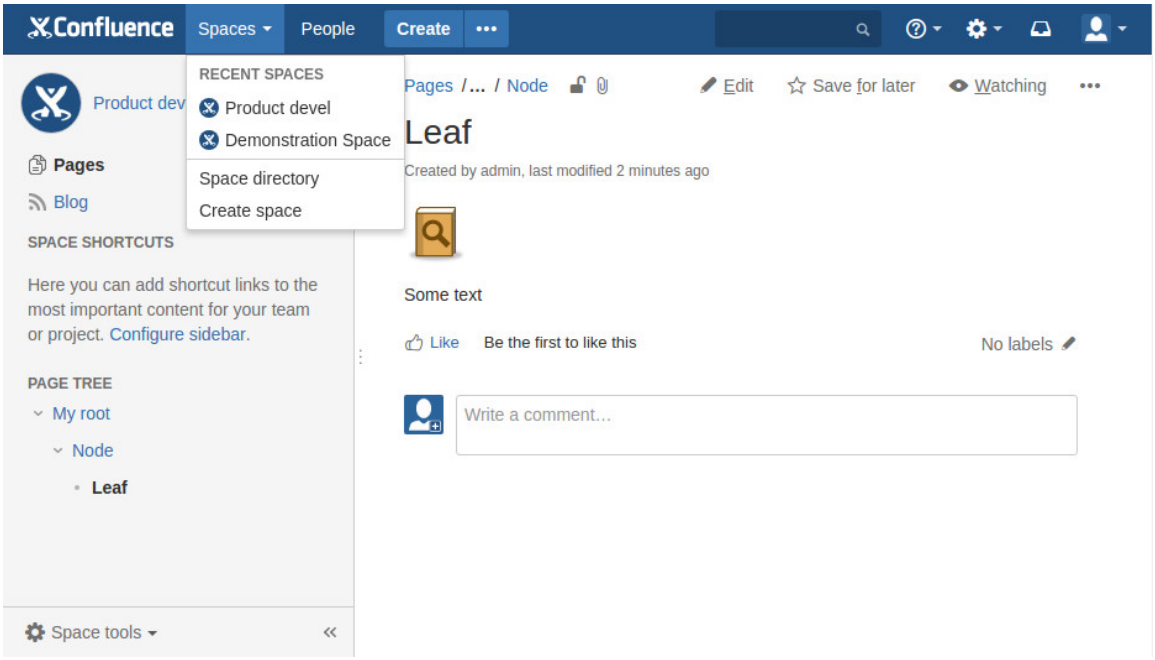
Kód 3.5: Ukázka použití knihovny Tesseract.NET.

```
using (var engine = new TesseractEngine(
  datapath: @"./tessdata",
  language: "eng",
  engineMode: EngineMode.Default)
) using (var img = Pix.LoadFromFile("img.tif"))
using (var page = engine.Process(img))
using (var iter = page.GetIterator()) {
    iter.Begin();
    do {
      // precteni slova
      var word = iter.GetText(PageIteratorLevel.Word);
    } while (iter.Next(PageIteratorLevel.Word));
}
```

3.4 Nástroj Confluence

Jedná se o nástroj společnosti Atlassian, jeho cílem je mj. zjednodušení týmového sdílení dokumentů. Tento nástroj poskytuje aplikační rozhraní ve formě REST[12]. Na obrázku 3.1 lze vidět jeho rozhraní.

[8] http://jocr.sourceforge.net/, https://linux.die.net/man/1/gocr
[9] http://jocr.sourceforge.net/api/)
[10] https://www.gnu.org/software/ocrad/
[11] https://github.com/tesseract-ocr/tesseract/
[12] https://developer.atlassian.com/confdev/confluence-server-rest-api



Obrázek 3.1: Webové rozhraní Confluence

3.5 Architektura REST

Zkratka REST znamená "Representational State Transfer"a nemá ustálený český překlad. Jedná se o bez-stavovou klient-server architekturu, která je definována šesti omezeními:

Jednotné rozhraní Definuje rozhraní mezi klienty a servery. Zjednodušuje a odděluje architekturu, což umožní nezávislý vývoj jednotlivých částí. Toto řešení však na druhou stranu zhoršuje efektivitu, neboť jsou data přenášena ve standardizované podobě na rozdíl od podoby, která je vyžadována požadavky aplikace. [12]

Bezstavovost Každý požadavek klienta na server musí obsahovat všechny potřebné informace. To umožňuje lépe monitorovat jednotlivé požadavky - ty obsahují vždy celou stavovou informaci. Na druhou stranu je tímto zvýšen objem přenášených dat. Díky bezstavovosti je pro systém jednodušší se zotavit z částečného selhání. Škálovatelnost je zlepšena, není třeba uchovávat zdroje pro užití mezi požadavky. [12]

Vyrovnávací paměť (Cache) Toto omezení vyžaduje, aby data v odpovědi byla označena, zda mohou, či nemohou být uložena ve vyrovnávací paměti. Vyrovnávací paměť zvyšuje výkon, avšak může způsobit použití neaktuálních dat. [12]

Klient-server Oddělení uživatelského rozhraní od úložiště dat; zvýší přenositelnost a škálovatelnost díky jednodušší implementaci serverových komponent. [12]

Vrstvený systém Vrstvený systém umožňuje, aby byla architektura složena z hierarchických vrstev, které mají přímý přístup pouze k nejbližší vrstvě. To omezuje celkovou komplexitu systému a umožňuje nezávislost na spodních vrstvách. Tímto postupem je také možné zlepšit škálovatelnost s využitím vrstev pro vyvažování zátěže. [12]

Kód na vyžádání Jedná se pouze o "volitelné omezení". Umožňuje, aby byla funkcionalita klienta rozšířena stažením a spuštěním kódu ve formě appletu či skriptu. [12]

RESTful HTTP

Implementace architektury REST s využitím HTTP bývá označována jako RESTful HTTP. Tuto implementaci nelze nazývat novou technologií, ale spíše definicí principů a omezení. [23]

Přístup ke zdrojům Rozhraní RESTful HTTP se skládá z HTTP metod, jako jsou GET, DELETE, PUT, či POST. Tyto metody musí být užity dle HTTP specifikace. Například použití metody GET k vytvoření či modifikaci objektu porušuje tyto principy. [23]

GET Slouží k získání dat. V případě využití systému cache je také možné vyžádat data pouze v případě změny. Tato metoda nemění stav serveru a je idempotentní. [23]

DELETE Odstraní data. Metoda je idempotentní. [23]

PUT Slouží k vytvoření či nahrazení dat na serveru s využitím ID spravovaného klientem. Metoda je idempotentní. [23]

POST Vytvoření dat s využitím automaticky spravovaných ID, či jejich částečnou aktualizaci. Metoda není idempotentní. [23]

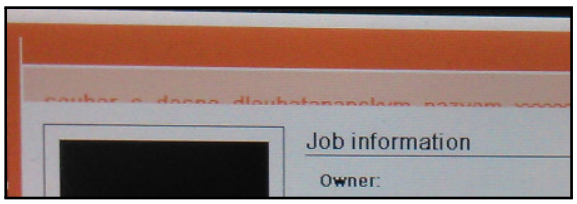
Kapitola 4

Popis vlastní práce

Tato kapitola se věnuje samotné kontrole zobrazení tlačítek a nahrávání fotografií do dokumentace. Mezi hlavní sekce patří například 4.3, která popisuje postup detekce tlačítek, na kterých je poté kontrolováno přetékání, což je popsáno v sekci 4.5. Nahrávání do dokumentace se věnuje sekce 4.7. Dosažené výsledky jsou popsány v sekci 4.8.

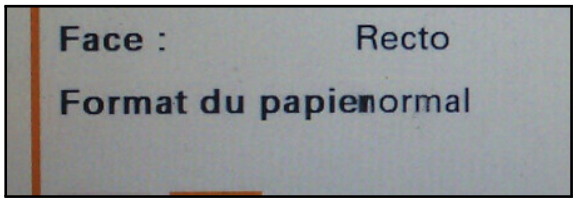
4.1 Definice problému

Společnost Y Soft Corporation nabízí tisková řešení. Jednou z věcí, kterou to obnáší, je úprava uživatelského rozhraní dotykového displeje tiskárny. K tomu je použito poskytované rozhraní tiskárny. Vstupem je definice zobrazovaných dat a samotné vykreslení je realizováno tiskárnou. Během vykreslování však mohou nastat problémy, při kterých dochází k ořízntutí či přetečení textů. Tato situace často nastává jen v některých z velkého množství jazykových mutací a je nákladné toto kontrolovat manuálně. Jelikož například přímé napojení na obrazový výstup tiskárny je komplikované a neuniverzální řešení, je zvolen způsob získávání obrazu pomocí focení obrazovky displeje. Na obrázcích 4.1 lze vidět některé z možných problémů.

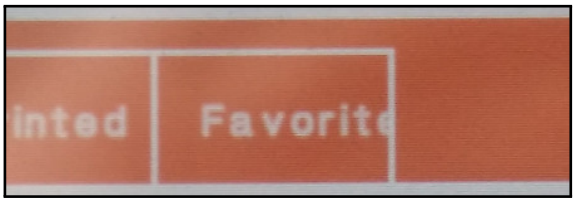


(a) Text je ořezán spodní hranou

(b) Text je zalomen a ořezán



(c) Text přesahuje do jiného textu



(d) Text je ořezán

Obrázek 4.1: Ukázky některých druhů nesprávného zobrazení

Vstupem této práce jsou fotografie, které jsou již patřičně upravené - oříznuté a zkosené. Kvalita těchto fotografií je však v současné době ať už více či méně ovlivněna rozmazáním,

moaré efektem a odleskem od povrchu displeje. Výstupem by měly být detekované chyby v zobrazení.

Cílem projektu je na těchto fotografií detekovat nesprávná zobrazení. Druhým úkolem je vybrat ze skupiny fotografií nejkvalitnější z nich a tu následně nahrát do dokumentace.

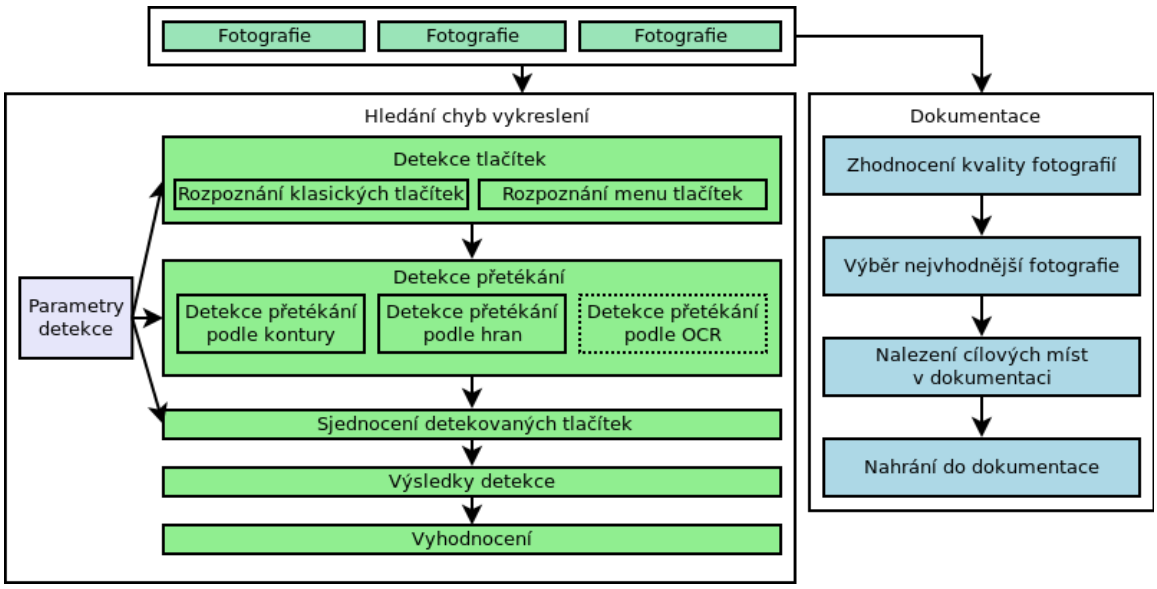
4.2 Koncepce řešení

Vstupem je obecně několik fotografií obrazovek. Ty mohou být následně buď nahrány do dokumentace, či zkontrolovány na přítomnost přetékajících textů.

V případě, že bude cílem nahrát fotografie do dokumentace, bude nejdříve provedeno zhodnocení kvality jednotlivých fotografií. Z těchto fotografií bude vybrána nejvhodnější fotografie, která bude nahrána do dokumentace. Tímto procesem se zabývá sekce 4.7.

Druhým možným postupem práce se vstupními fotografiemi je kontrola zobrazení. Projekt se zaměřuje na typ defektu, kdy text přetéká přes okraje tlačítka, nebo je tlačítkem oříznut. V první fázi proběhne nalezení tlačítek v obraze - 4.3. Poté bude zkontrolováno, zda na zobrazených tlačítkách nedochází k přetékání. V některých případech dochází pouze k oříznutí. V dalším textu bude oříznutí považováno za formu přetékání. Této problematice se věnuje sekce 4.5. Některá nalezená tlačítka se mohou překrývat. To může být způsobeno například chybnou detekcí či detekcí tlačítka více metodami. Toto bude řešeno v sekci 4.6. Poslední fází je vyhodnocení výsledků, které je popsané v sekci 4.8. Parametry jednotlivých kroků jsou dostupné z jednoho místa - z třídy **DetectionOptions**. Mezi tyto parametry patří například minimální a maximální rozměry tlačítek, které budou rozpoznány. V případě získávání špatných výsledků lze úspěšnost analýzy často zvýšit jen pomocí modifikace těchto parametrů.

Celý postup je znázorněn schématem 4.2. Jsou zde znázorněny dva způsoby zpracování fotografií, které jsou prakticky nezávislé na sobě.



Obrázek 4.2: Blokové schéma projektu

4.3 Detekce tlačítek

Tato sekce popisuje několik zvažovaných postupů detekce. Následně také uplatnění zvoleného postupu.

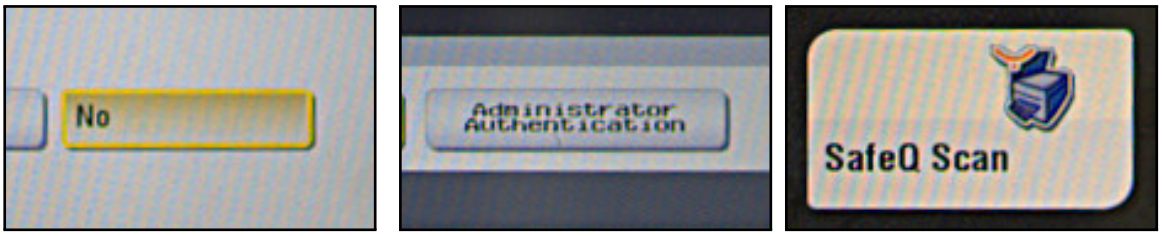
Definice termínu tlačítko

Za účelem detekce tlačítek je vhodné nejdříve definovat, co znamená `tlačítko` v kontextu této práce. Neexistuje bohužel žádná objektivní specifikace a následující definice je odvozena pouze na základě subjektivního pozorování několika referenčních fotografií.

Obdélníkový tvar Většina tlačítek má obdélníkový tvar a mají větší horizontální než vertikální rozměr. Existují však i tlačítka čtvercová - ta obvykle obsahují jen písmeno, nebo obrázek.

Zakulacené rohy Různý počet rohů tlačítka může být zakulacený. V extrémních případech je svislou hranou tlačítka půl-kružnice. Kulatá tlačítka nebudou cílem této práce.

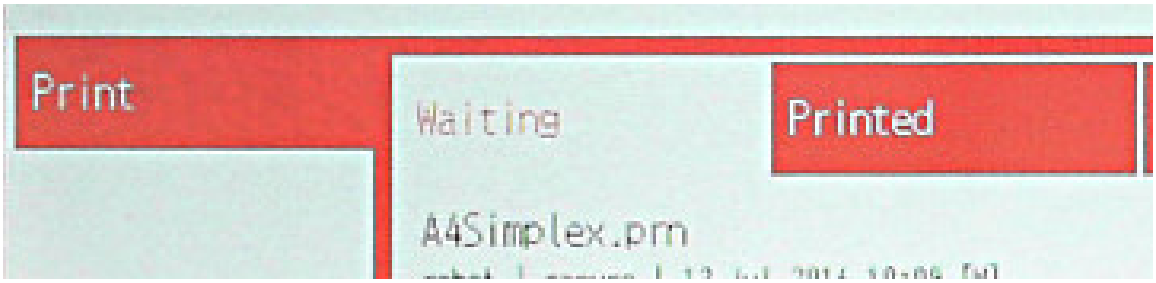
Text Každé tlačítko obsahuje text[1] - 4.3a, v některých případech i víceřádkový - 4.3b. Spolu s tímto textem se může v tlačítku objevit i ikona - 4.3c.



(a) Jednoduchý text (b) Víceřádkový text (c) Ikona

Obrázek 4.3: Typy tlačítek v závislosti na textu

Součást menu Některá tlačítka jsou součástí horizontálního či vertikálního menu. Od klasického vzhledu tlačítka se mohou lišit tím, že chybí jedna ze čtyř hran tlačítka. Taková tlačítka však lze očekávat pouze na horní, či levé části obrazovky.



Obrázek 4.4: Ukázka tlačítek v menu

[1] Tlačítka pouze s ikonou není nutné rozpoznat.

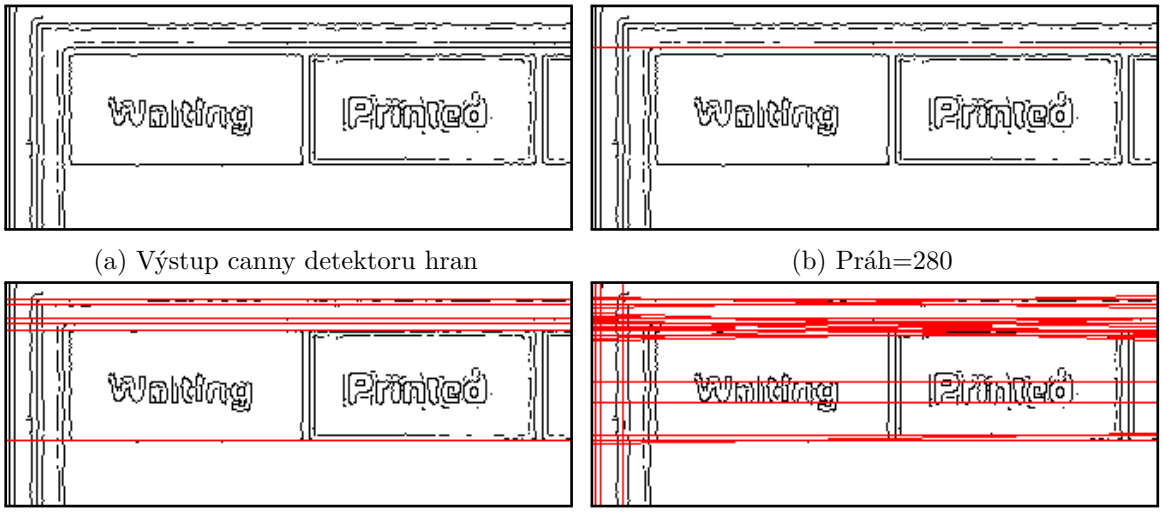
Oba typy tlačítek lze vidět na obrázku 4.4. Tlačítko „Print“ reprezentuje tlačítko vertikálního menu. Jeho pravá strana není nijak ohraničena a přímo se pojí k oranžovému rámečku. Tlačítko horizontálního menu je reprezentované tlačítkem „Waiting“. Okraje tlačítka tvoří ze dvou stran oranžový rámeček, z pravé strany je okraj tvořen tlačítkem „Printed“ a spodní okraj není nijak ohraničen.

Volba základního přístupu pro detekci tlačítek

Tato sekce popisuje potenciální postupy detekce tlačítek.

Detekce na základě prahování Jako nejjednodušší způsob, jak oddělit objekty, tedy tlačítka od pozadí, se může jevit prahování. Prahování je možné provádět na základě různých vlastností pixelů. V případě obrázku využívající RGB barevný model to může být například intenzita červené, zelené či modré. V obrázku využívajícím pouze odstíny šedi to může být například intenzita pixelu. Zajímavé výsledky lze také získat po převedení obrazu do jiných barevných modelů, jako je například HSV, a prahovat například podle barevného tónu. Pro potřeby hledání tlačítek však není možné najít vhodné parametry prahování a to především z důvodu různých barev a stylů tlačítek. Pokud bychom předem věděli, jaký typ fotografie bude zpracováván, využití tohoto postupu by mohlo být jednoduché a rychlé.

Detekce tlačítek na základě hran Tato sekce zvažuje využití Houghovy transformace pro detekci tlačítek. To je však problematické z důvodu nekvalitního obrazu a neznalosti velikosti tlačítek. V praxi je problém správně nastavit potřebný práh - a určit, zda skupina pixelů tvoří hranu, či ne. Dalším problémem je krátká vzdálenost mezi tlačítky, což způsobuje problémy s rozlišením hrany rozpoznávaného tlačítka od hrany sousedního tlačítka.



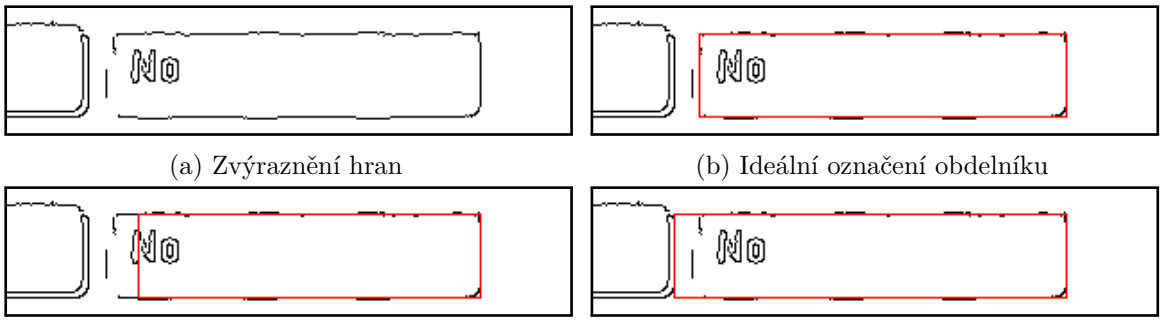
(a) Výstup canny detektoru hran (b) Práh=280

(c) Práh=150 (d) Práh=90

Obrázek 4.5: Nalezené přímky(červeně) v obraze hran pomocí Houghovy transformace

Na obrázku 4.5 vidíme výsledky Houghovy transformace pro různé prahy - hodnota akumulátoru pro danou přímku dosahuje alespoň nějaké hodnoty. Cílem by mělo být nalezení 4 přímek, které tvoří tlačítko. V praxi by bylo vhodné volit mírnější práh pro vertikální přímky, které jsou kratší než přímky horizontální.

V obrázku 4.5b byla hranice nastavena příliš přísně a hrana tlačítka nebyla vůbec nalezena. V obrázku 4.5c byly nalezeny téměř všechny horizontální hrany. Zvolíme-li ještě mírnější práh 4.5d, metoda nalezne přímky i v místech, kde se reálně nenachází. Toto je jeden z problémů Houghovy transformace. V případě, že neznáme délku hledaných hran, nemůžeme určit vhodnou hodnotu prahu. Kdybychom měli na obrazovce pouze jedno tlačítko, bylo by možné najít 4 maxima v Houghově prostoru.



(a) Zvýraznění hran

(b) Ideální označení obdelníku

(c) Příliš nízký práh

(d) Příliš vysoký práh

Obrázek 4.6: Ukázka problému s výběrem hran

V našem případě bohužel bývají tlačítka příliš blízko u sebe a i v případě použití Houghovy transformace s posuvným oknem se tento problém nevyřeší. Představme si, že posuvné okno zobrazuje pouze jedno celé tlačítko a blízké okolí. Hledá-li se levá strana tlačítka, tak v případě nízkého prahu bude nesprávně za hranu označen například kousek textu. Naopak pokud by byl zvolen práh příliš vysoko, za levou hranu tlačítka bude označena až pravá strana přilehlého tlačítka, které se částečně nachází ve výběru.

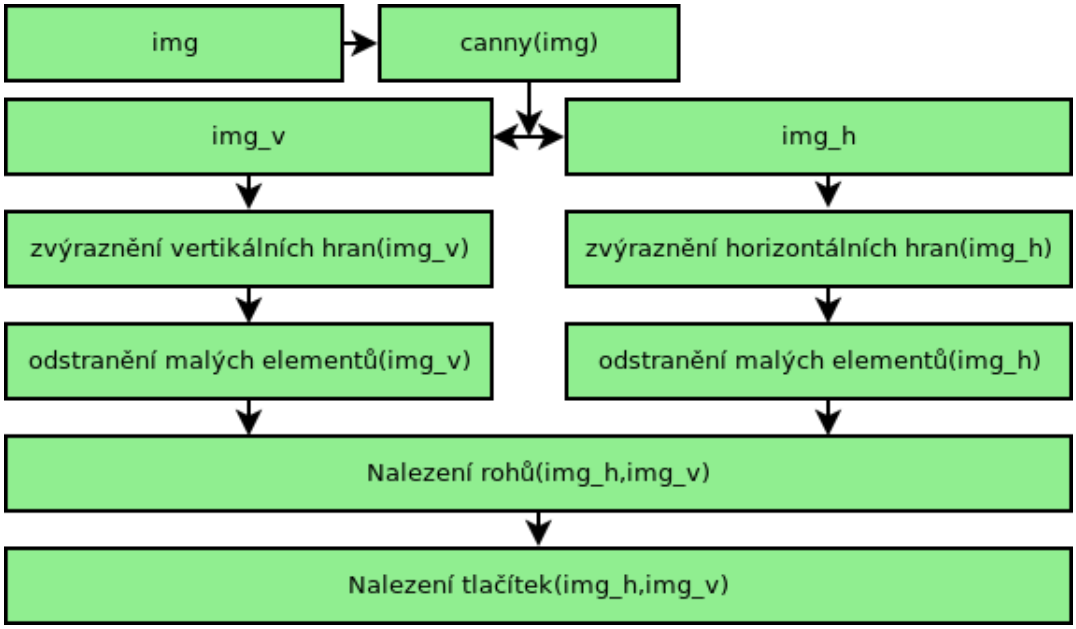
Na obrázku[2] 4.6 vidíme, jaké budou výsledky při použití různých prahů. Pokud algoritmus použije i příliš nekvalitní hranu, může dojít k označení části textu jako hrany. Pokud algoritmus bude hledat velmi kvalitní hranu, může nalézt nesprávnou hranu. Lze předpokládat, že využití jiné metody založené pouze na hranách by trpělo podobnými problémy.

Detekce tlačítek pomocí rohů Tento postup vychází z předpokladu, že každé tlačítko se kromě ze čtyř na sebe kolmých úseček skládá ze čtyř (i zakulacených) rohů. Tyto rohy vzniknou právě v místech, kde se horizontální hrany tlačítka propojují s hranami vertikálními, a naopak.

Ve schématu 4.7 lze vidět, že ze vstupního obrázku jsou vytvořeny dvě kopie, které jsou dále zpracovávány odděleně. Výhoda tohoto řešení je, že poté kromě samotných rohů získáme také vertikální a horizontální hrany, což bychom nezískali, kdybychom užili samotný detektor rohů. Detekované rohy se nacházejí na průnicích obou obrázků. Každý takto vzniklý průnik vytvoří rohový pixel. Po transformaci hrany tlačítka hranovým detektorem však často nezískáme pouze jednu dokonalou hranu. Proto je za roh považováno i jisté okolí průniku. Ve výsledku se tato okolí v určitých místech spojí - celou tuto oblast poté považujeme za roh. Pro každou z takto vzniklých oblastí také určíme čtveřici, která reprezentuje informaci, z kterých čtyř stran vycházejí hrany.

Představme si výsledné tlačítko jako n-tici 4 rohů (r_{lh}, r_{ph}, r_{ld}, r_{pd}). Typ rohu tedy můžeme definovat pomocí hran, které z nich vychází - například pravý dolní roh má hrany směrem nahoru a doleva. Každá oblast rohu má určitou výšku a šířku. Tyto rozměry poté

[2] Obrázek byl pro názornost ručně upraven.



Obrázek 4.7: Blokové schéma algoritmu detekce podle rohů

určují i velikost prohledávaných oblastí pro nalezení nejbližších rohů, tedy šířku vertikálních oblastí a výšku u horizontalních oblastí. Prohledávání nejbližších rohů vždy probíhá buď pouze vertikálně, či horizontálně - tedy pokud máme levý horní roh, poté nejbližší pravý horní roh hledáme pouze směrem doprava.

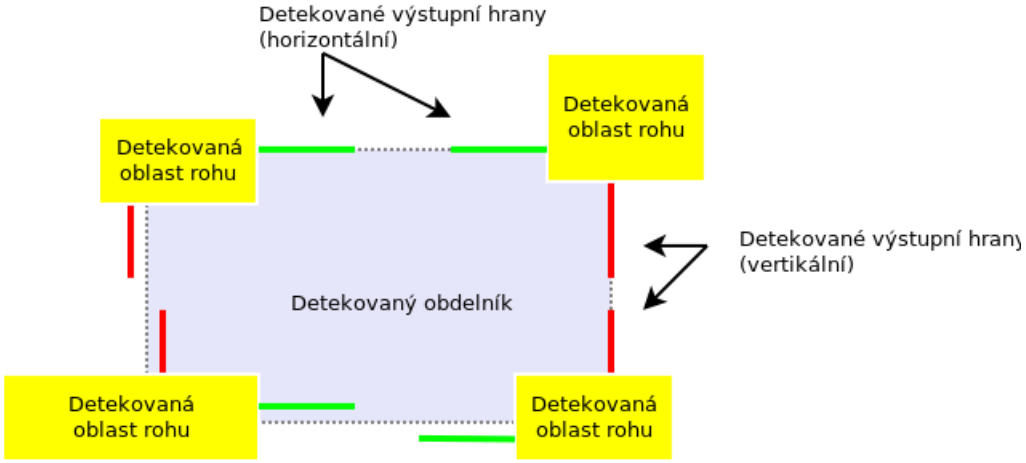
Budeme-li ignorovat tlačítka, v kterých dochází k přetékání textu, je hledání tlačítek v množině rohů poměrně snadnou záležitostí. Pro všechny levé horní rohy r_{lh} najdeme nejbližší pravý horní roh r_{ph} a nejbližší levý dolní roh r_{ld}. Poté získáme množinu potenciálních pravých dolních rohů R_{pd} jako nejbližší pravý dolní roh rohu r_{ph} a nejbližší pravý horní roh rohu r_{ld}. Pokud množina R_{pd} obsahuje pouze jeden prvek r_{pd}, můžeme (r_{lh}, r_{ph}, r_{ld}, r_{pd}) prohlásit za tlačítko.

Poslední fázi prezentované metody by bylo možné vylepšit, aby byla robustnější i vůči tlačítkům, která jsou poškozena přetékajícím textem.

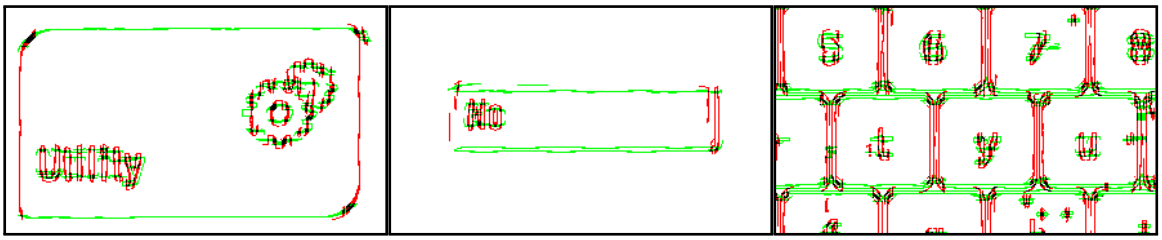
Hlavní problém tohoto postupu se však ukrývá již na začátku. Tlačítka mají často zaoblené rohy. V některých případech to způsobí pouze nesprávnou detekci velikosti tlačítka. V jiných případech se však bohužel často stává, že i pokud jsou k dispozici poměrně kvalitní hrany, tak průniky těchto hran v rozích chybí.

Dalším problémem je detekce vycházejících hran. Tlačítka často nejsou pouhé ploché obdélníky, ale mají plastický tvar. Toto způsobuje, že i pokud máme pravý roh tlačítka, vedou z něj často hrany směrem doprava, které jsou způsobeny plasticitou tlačítka. Filtrování takových hran je poměrně komplikované, neboť všechny hrany jsou poškozeny nějakým typem šumu a je poměrně nesnadné rozlišit mezi hranou potřebnou a hranou určenou k ignoraci.

Na obrázku 4.9a lze vidět, že i některé rohy zaoblených tlačítek jsou takto detekovány. Pokud bychom však tyto rohy chtěli propojit, nalezený obdélník by byl menší než reálné tlačítko. Obrázek 4.9b zobrazuje tlačítko, kde i s poměrně kvalitními hranami chybí jeden z rohů. Také lze vidět, že z pravého dolního rohu vybíhá hrana částečně doprava, což by mohlo způsobit nesprávné propojení okolních hran. Poslední obrázek 4.9c zobrazuje



Obrázek 4.8: Popis tlačítek pomocí rohů



(a) Tlačítko se zaoblenými rohy (b) Tlačítko s chybějícím rohem (c) Skupina tlačítek

Obrázek 4.9: Zobrazení zelených horizontálních hran, červených vertikálních a jejich průniku

skupinu tlačítek, u kterých se jednotlivé rohy přilehlých elementů propojí, což způsobuje další komplikace při detekci obdélníků.

Detekce tlačítek s využitím regionální segmentace Oproti metodám vycházejícím pouze z hran obrazu se jako poměrně slibné řešení jeví využití regionální segmentace. K tomuto účelu je využita varianta mean-shift segmentace, při které je obraz před-filtrován pomocí mean-shift metody a následně je provedena segmentace opakovaným využitím semínkového vyplňování[3]. Knihovna OpenCV nenabízí implementaci klasické mean-shift segmentační metody. Výše zvolený způsob je výhodný z toho důvodu, že je možné využít již implementovaných algoritmů pro filtraci a semínkového vyplňování, které se nachází v nativním kódu knihovny OpenCV. Pro využití klasické mean-shift segmentace by bylo třeba zvýšit množství využívaných knihoven, či implementovat algoritmus v jazyce C#, který je obecně méně efektivní než nativní kód.

V prvním kroku se provede nejprve mean-shift před-filtrování, které způsobí vyhlazení šumu se zachováním hran. Vstupní obraz je také zpracován pomocí Canny detektoru hran. Na hrany je ještě aplikována dilatace a eroze, obecně s různými parametry. To pomáhá zacelit hrany tlačítka a zvýšit šanci úspěšné segmentace. Výsledek této operace je viditelný například při rozpoznávání tlačítka zobrazeném na obrázku 4.10a. Samotné tlačítko má složitý rámeček, který způsobí, že výsledkem jsou dvě hrany detektoru hran - vnitřní a

[3] https://github.com/opencv/opencv_attic/blob/master/opencv/samples/cpp/meanshift_segmentation.cpp

vnější. Ve výsledku však tyto hrany správně neuzavírají oblast tlačítka. V některých případech může být barva tlačítka podobná barvě pozadí, na kterém je tlačítko umístěno a při segmentaci je poté segment tlačítka nesprávně považován za součást segmentu pozadí. V ideálním případě je tedy vhodné mít oblast tlačítka ohraničenou hranami. Na obrázcích 4.10a a 4.10b lze v červeně zvýrazněné oblasti vidět, jak operací uzavření dojde k propojení těchto hran. Naopak v modré oblasti k tomu bohužel nedošlo. Řešením by bylo zvolení většího jádra pro výše použité morfologické operace. To však na druhou obecně zvyšuje riziko spojení nesouvisejících hran, které spojeny být neměly - například spojení obrysu tlačítka s vnořeným textem.

(a) Výřez fotografie obsahující tlačítko

(b) Stav před hran před provedením uzavření

(c) Stav po provedení uzavření

Obrázek 4.10: Provedení operace uzavření na hranový obraz

V tuto chvíli je k dispozici obraz vyfiltrovaný od mírných nerovností a obraz tvořený hranami původního obrazu. Na vyhlazeném obrazu je provedena segmentace spojením oblastí s podobnou barvou. OpenCV poskytuje funkci `FloodFill()`, která kromě zdrojového bodu pro semínkové vyplňování a odchylky barvy pracuje také s osmibitovou maskou, která má velikost vzorového obrazu + 1-pixelového okraje.

Každému pixelu zdrojového obrázku tedy připadne jeden pixel masky. Osmibitovou masku si však lze představit pouze jako binární, jenž reprezentuje dva stavy - `je možné daný pixel vyplňovat` a `není možné daný pixel ještě vyplňovat`. Funkce pro semínkové vyplňování používá tuto masku jednak jako omezení svého vyplňování, ale také jako jeden ze svých výstupu. Během vyplňování jsou hodnoty masky v daných pixelech upraveny tak, aby při opakovaném vyplňování nebyly pixely vyplněny podruhé. Průběžné kroky tohoto postupu jsou znázorněny na obrázku 4.11.

Každé volání funkce **FloodFill()** tedy vyplní určitý region a vyplní odpovídající oblast v masce. Tato část masky je poté analyzována. Analýza zařadí část masky do jedné ze tří kategorií, které lze popsat takto:

- maska neodpovídá tlačítku
- maska odpovídá tlačítku
- část masky odpovídá tlačítku - jedná se o masku menu

Samotné rozpoznání tlačítka však také není jednoduchým úkolem, neboť vlastnosti tlačítka nejsou jasně definované. Dalším problémem jsou defekty fotografie.

Rozpoznání tlačítek ze segmentů obrazu

Je k dispozici segment, který potenciálně odpovídá tlačítku. Nyní je třeba podle jeho vlastností rozhodnout, zda se o tlačítko jedná. Konkrétní parametry následujících kontrol zde nebudou popsány, neboť vstupují do procesu externě a mohou se měnit.

(a) Originální obrázek (b) Hranový obraz

(c) Vyfiltrovaný obraz (d) Barevně odlišené segmenty obrazu

Obrázek 4.11: Znázorněný postup segmentace

Kontrola rozměrů tlačítka Některé útvary mohou z hlediska tvaru splňovat vlastnosti tlačítka. Mezi tyto útvary mohou patřit rámečky kolem oblastí v obraze, či malé oblasti uvnitř písmen, či v šumu. Proto je třeba kontrolovat také rozměry těchto oblastí. Jedním kritériem pro filtraci dle rozměrů je samozřejmě odstranění elementů s příliš malou šířkou či výškou. Tato kontrola také využívá velikost vstupního obrazu. Rozměry tlačítka nesmí překročit jistou procentuální část těchto rozměrů. V dostupných referenčních datech se často vyskytovala tlačítka[4], která měla horizontální rozměry blízké rozměrům fotografie. Vertikální rozměry tlačítek však často zabírají menší část obrazovky. Požadavek je také na poměr stran. V referenčních datech se vyskytovala pouze tlačítka s delší vodorovnou hranou než vertikální a také čtvercová tlačítka. Nelze však ignorovat všechna potenciální tlačítka, která mají výšku delší než šířku, neboť samotná detekce není absolutně přesná a získané rozměry a pozice se mohou mírně lišit od reálných. Je tedy kontrolován maximální poměr mezi šířkou a výškou. Filtrace tlačítek dle rozměrů je užitečná také v případě, kdy fotografie obsahuje vnořené obdélníky. Vnořená tlačítka nejsou validním uživatelským prvkem a detektor musí rozhodnout, který z vnořených prvků je tlačítkem. Toto rozhodnutí je obecně poněkud problematické, neboť samotná tlačítka mohou vypadat různě a obsahovat vnořené ikony připomínající obdélníky. Filtrace na základě rozměrů jednoduchým způsobem pomáhá vyhnout se velké části těchto problémů.

Kontrola vyplněné plochy Některá tlačítka mají například komplexní typy rámečků - viz 4.10. Regionální segmentací poté vzniknou oblasti, které na základě vnějších rozměrů a tvarů splňují požadavky tlačítka. Tyto elementy však mají prakticky velmi malou plochu vzhledem ke svým rozměrům. Základní kontrola těchto elementů probíhá podle množství

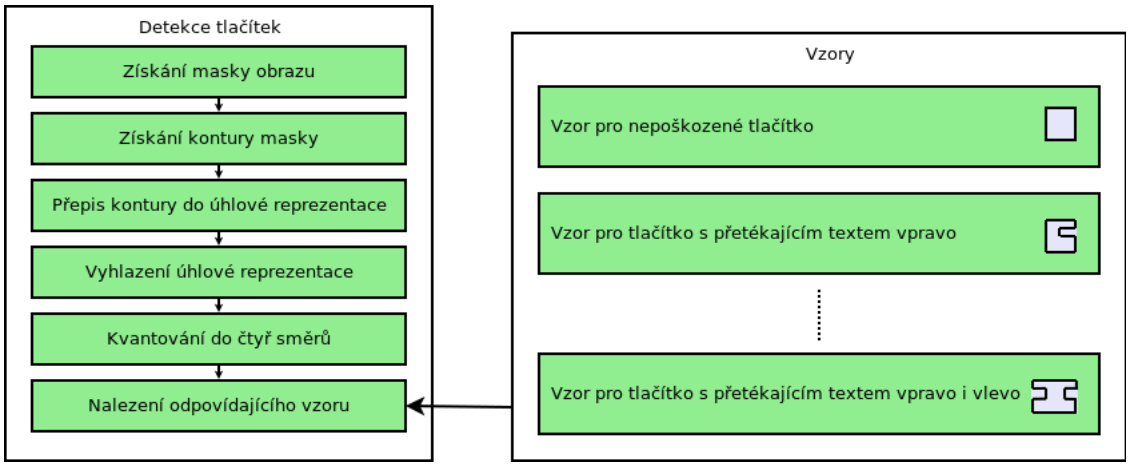
[4]Či vlastnostmi podobné prvky, u kterých je také vhodné provádět kontrolu

vyplněných pixelů, jejichž počet je získán jako výsledek volání funkce **FloodFill()**. V případě, že element projde tímto filtrem, je provedena ještě druhá kontrola, která spočívá ve využití eroze na následné kontrole, zda element nebyl úplně odstraněn.

Na obrázku 4.12 lze vidět výsledek segmentace, jehož jednotlivé segmenty liší svou barvou v odstínech šedi. Podstatný je červeně zvýrazněný segment. Velikost jeho obklopujícího obdélníku má podobné rozměry jako tlačítko použité v obraze. Prakticky se však jedná jen o jeden z rámečků.

Obrázek 4.12: Ukázka segmentu vzniklém rámečkem

Kontrola tvaru kontury Pokud element splňuje výše uvedené podmínky, které jsou relativně rychle realizovatelné, proběhne kontrola na tvar kontury segmentu.



Obrázek 4.13: Blokové schéma rozpoznávání tlačítek dle jejich kontur.

Rozpoznání signatur tlačítka v masce zaštiťuje třída **ContourButtonDetector**, která obsahuje několik nadefinovaných vzorů, jak může vypadat tlačítko - tedy jeden vzor pro validní tlačítko a několik dalších vzorů pro tlačítka obsahující přetékající text. Důležitá je metoda **Match()**, která určí, zda a o jaké tlačítko se jedná.

Interně tato metoda převede masku segmentu na popis její kontury. Tento popis si lze představit jako kruhový seznam dvojic (směr, délka kontury). Na tento popis se poté postupně pokouší aplikovat jednotlivé vzory.

Jak vypadá popis kontury si lze ukázat na ilustračních obrázcích 4.14. Obrázek 4.14a znázorňuje červené tlačítko bez textu a jeho konturu vyznačenou černě. Tuto konturu skládající se ze čtyř stran lze popsat jako posloupnost `doleva, dolů, doprava a nahoru`. V případě tlačítka s přetékajícím textem 4.14b dojde k narušení jeho vnějšího obrysu, jak je znázorněno na obrázku 4.14c. Po aplikaci filtru získáme konturu znázorněnou na obrázku 4.14d. Tu lze popsat posloupností `doleva, dolů, doprava, nahoru, doleva, nahoru, doprava a nahoru`.

(a) (b) (c) (d)

Obrázek 4.14: Znázornění kontur v různých případech (a) Tlačítko bez textu (b) Tlačítko s přetékajícím textem (c) Kontura tlačítka s přetékajícím textem (d) Vyfiltrovaná kontura

Získávání popisu kontury z masky Na vstupní masku je aplikována OpenCV funkce **FindContours()**, jež vrátí seznam pozic pixelů, které tvoří hrany elementu.

Vytváření jakýchkoliv závěrů na základě tohoto seznamu však není triviální záležitost, neboť získané hrany často obsahují jisté zkreslení a nejsou dokonale rovné. I vychýlení hrany o jeden pixel by způsobilo, že bychom jednu vertikální hranu viděli jako hrany tři: vertikální, horizontální a opět vertikální. Pokud chceme hledat odpovídající vzory v tomto popisu, je třeba provést vyfiltrování nepřesností v těchto obrysech.

Prvním krokem je převedení seznamu pixelů na kruhový seznam úhlů mezi těmito pixely. Tyto úhly vždy popisují, v jakém směru následuje další pixel v řadě. Budeme-li tedy mít dokonalou horizontální hranu elementu, tak všem bodům kromě posledního bude přiřazen stejný úhel.

Jako nejvhodnější filtrace se jevilo použití 1D konvolučního filtru, který by výsledný úhel v daném bodě spočítal jako průměr úhlů v jeho blízkém okolí.

Tabulka 4.1: Varianta možného 1D filtru

1	**1**	**1**	**1**	**1**	**1**

V případě použití tohoto filtru však nastává problém, neboť pro hodnoty reprezentující úhel nelze použít klasický aritmetický průměr. Proto je pro průměr úhlů využíván vzorec [5]:

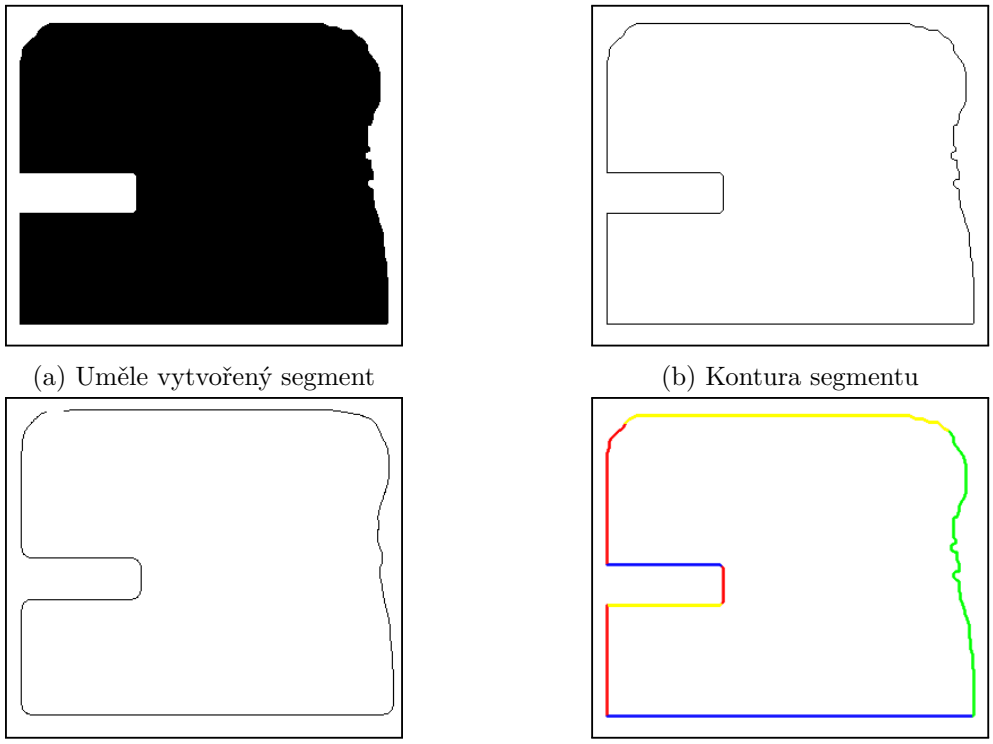
$$\alpha = \arctan(\sum_{i=1}^{n} \sin(x_i), \sum_{i=1}^{n} \cos(x_i)) \tag{4.1}$$

Poté, co proběhne filtrace těchto úhlů, provede se kvantování úhlů do čtyř směrů - `nahoru`, `dolů`, `doprava` a `doleva`.

Na obrázku 4.15a lze vidět uměle vytvořený segment pro vizualizaci získání popisu kontury. Dále, na obrázku 4.15b lze vidět jeho získaná kontura, která je reprezentovaná jako seznam pozic pixelů. Ten byl převeden do seznamu úhlů, vyfiltrován a vykreslen, jak lze vidět na obrázku 4.15c. Přerušení kontury je způsobeno pouze vizualizací po vyfiltrování, kdy poslední vykreslovaný pixel nenavazuje na první vykreslovaný pixel. Jedná se pouze o vlastnost při této vizualizaci. Vyfiltrované úhly použité v obrázku 4.15c byly kvantovány do čtyř směrů, které byly přiřazeny původním pixelům kontury. To lze vidět barevně znázorněné v obrázku 4.15d. Žlutá reprezentuje směr vlevo, červená dolů, modrá vpravo a zelená nahoru. Lze vidět, že mírné nerovnosti v obrysu byly ignorovány, zatímco ty výrazné byly zachovány.

Hledání vzorů v popisech kontur Odpovídající vzory je třeba nějakým způsobem reprezentovat. Vzory klasických tlačítek implementují rozhraní **IContourPattern**. Toto

[5] http://catless.ncl.ac.uk/Risks/7.44.html#subj4



(a) Uměle vytvořený segment

(b) Kontura segmentu

(c) Vyfiltrovaná kontura

(d) Zařazení hrany do směru

Obrázek 4.15: Ukázka filtrace kontury a kategorizace pixelů

rozhraní se zavazuje poskytnout 2 informace - typ poškození tlačítka a vzor odpovídajících úseků kontur.

Vzor tlačítka je reprezentován:

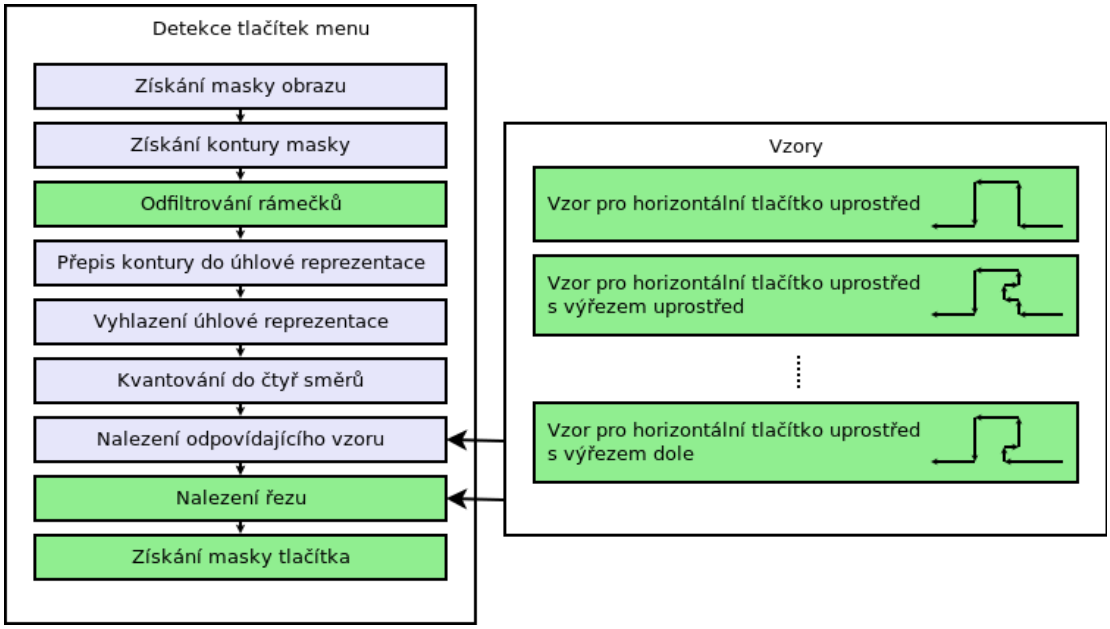
- seznamem očekávaných směrů hran
- požadovaným rozsahem délek těchto hran
- validačním pravidlem

V případě detekce klasických tlačítek se kontroluje shoda všech úseků masky s odpovídajícím vzorem, neboť popisujeme celý obvod masky. Validační pravidlo je kontrolováno až v případě, kdy se podaří nalézt shodu mezi odpovídajícími segmenty. Lze si jej představit například v situaci, kdy vzor odpovídá tlačítku s přetékajícím textem vlevo i vpravo a následné validační pravidlo vynutí, že přetékání textu na levé straně musí být ve stejné vertikální pozici(řádku) jako na straně pravé. Takovéto omezení nelze vynutit pouze nezávislými omezeními na délky segmentů.

4.4 Detekce tlačítek menu

Detekce tlačítek menu probíhá podobným způsobem jako u případu klasických tlačítek. Situace je však komplikovanější, neboť tlačítko nelze rozpoznat pouze pomocí čtyř jeho stěn.

Blokové schéma 4.16 kromě samotného postupu znázorňuje společné části s postupem pro klasická tlačítka. Sdílené bloky jsou reprezentovány modrou barvou.



Obrázek 4.16: Blokové schéma rozpoznávání tlačítek menu

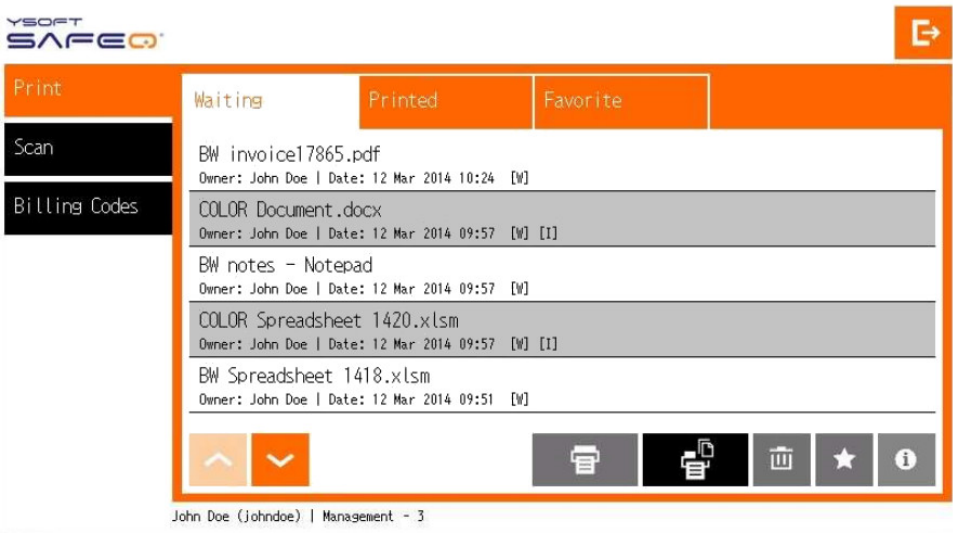
Vstupní obraz je zpracováván dvakrát. Pokaždé s jiným způsobem filtrování. Jedna filtrace pouze odstraní přebytečné hrany pomocí operace otevření. Druhý způsob filtrace nejdříve odstraní rámečky (pomocí operace otevření s větším jádrem než v případě prvního filtru) a poté vyplní díry operace uzavření.

Obrázky 4.18 znázorňují dva případy segmentů menu. Ve variantě 4.18b se snažíme detekovat tlačítko „Waiting“. Jedná se o variantu menu, samotný segment stránky obsahuje kromě části s tlačítkem také rámeček, který ohraničoval ostatní tlačítka menu. Hledáme-li tlačítko na základě vnějšího obrysu, nastaly by dva možné scénáře. Buď by žádné tlačítko nebylo nalezeno, neboť rámeček by nesplňoval limity pro velikosti tlačítka, nebo by byla detekována jako tlačítko celá horní část i s rámečkem. Proto bylo provedeno otevření, které ve fázi eroze odstraní tyto rámečky a ve fázi dilatace „opraví“ zbylé zerodované masky.

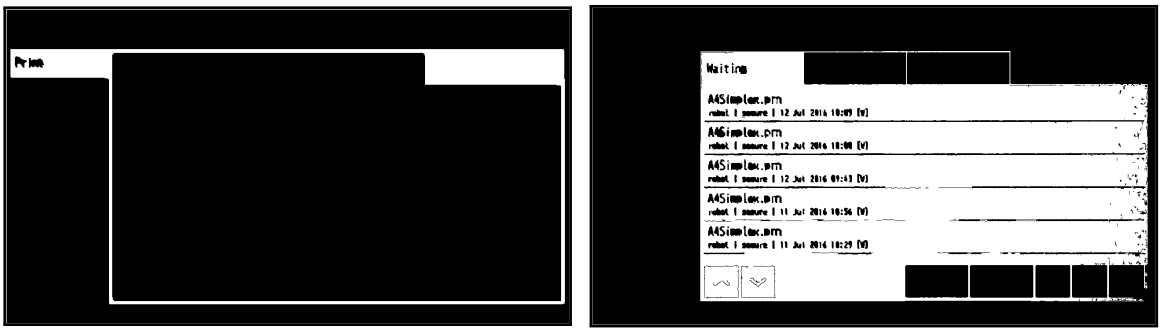
Ve variantě 4.18a vidíme tlačítko „Print“ jako výběžek ze segmentu, který je tvořen prakticky jen rámečkem. Odstraněním tohoto rámečku bychom získali dva samostatné obdélníky. Levý obdélník bychom sice mohli detekovat podobným způsobem jako klasická tlačítka, ale pravý bychom detekovali navíc.

Hledání vzorů menu tlačítek Rozpoznávání vzorů v popisech kontur probíhá podobně jako u klasických tlačítek, v některých částech se však liší. Vzory menu tlačítek implementují rozhraní **IMenuContourPattern**. Oproti vzoru pro klasické tlačítko je tímto rozhraním poskytován také typ tlačítka a dva body pro provedení řezu - bude popsáno níže.

Hlavním rozdílem je, že vzory nepopisují celý obvod masky, nýbrž pouze výstupek způsobený tlačítkem + nezbytné okolí. Je to výhodné například v situaci znázorněné na obrázku 4.18b. Přestože oblast tlačítka horizontálního menu obsahuje poměrně čisté obrysy, o spodní části masky segmentu se to říci nedá. Není tedy důvod, aby se rozpoznání tlačítka nepodařilo z důvodu kladení zbytečných požadavků na nerelevantní hrany.



Obrázek 4.17: Použití menu tlačítek „Waiting“ a „Print“



(a) 1. Varianta menu

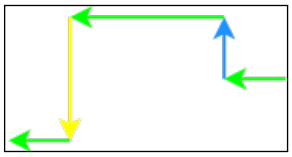
(b) 2. varianta menu

Obrázek 4.18: Zobrazení masek menu

Jako v případě klasických tlačítek je definováno několik vzorů, které mají najít různé případy. Některé vzory očekávají, že hrany tlačítka budou bez přetékání, jiné očekávají poškození tlačítka

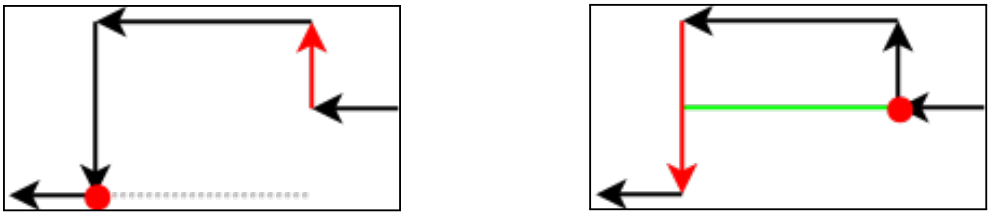
Hledáme-li například tlačítko horizontálního menu, lze předpokládat segmenty s orientací doleva, nahoru, doleva, dolů a doleva. Na každý tento segment mohou být aplikovány požadavky jak z hlediska minimální tak maximální délky. Takto samostatně kladené požadavky na délky však stále nemohou zajistit, aby vzor nebyl rozpoznán na nesprávném místě. Mohlo by tak dojít například k nalezení tlačítka, jež má jednu svislou stranu dvakrát delší než tu první. Přestože obě strany nezávisle na sobě splňují požadavky, jako celek požadavky nesplňují - neboť spodní hrany nejsou v jedné rovině. To lze vidět na obrázku 4.19. Na rozdíl od detekce klasických tlačítek je poměrně často využívána validační kontrola po nalezení shody segmentů se vzorem. Tyto kontroly samozřejmě musí počítat s nějakými vhodnými odchylkami.

Nalezneme-li shodu vzoru menu tlačítka, stále ještě neznáme přesné umístění tohoto tlačítka. Za účelem označení prostoru tlačítka probíhá jeho oddělení od zbytku masky. To probíhá pomocí řezu mezi dvěma body poskytovanými vzorem tlačítka, pomocí kterých proběhne řez - vždy buď horizontální, nebo vertikální.



Obrázek 4.19: Zobrazení nekorektního obrysu tlačítka

Hledání dvojice bodů probíhá určením prvního bodu a druhý bod je vybírán z protější hrany. Je vybírán takový bod, který sdílí stejnou x-ovou souřadnici, jedná-li se o vertikální řez, či y-ovou souřadnici, jedná-li se o řez horizontální. V případě, že se nenalezne odpovídající druhý bod, je hledání opakováno z druhé strany. Na získané části masky lze poté provádět analýzu přetékání apod.



(a) Neúspěšný řez

(b) Úspěšný řez

Obrázek 4.20: Zobrazení řezu masky

Obrázek 4.20 znázorňuje průběh hledání bodu řezu, který by měl oddělit horizontální tlačítko od spodní části masky. Jsou zde znázorněny rozpoznané segmenty tlačítka a červeně znázorněný bod, kterým bude probíhat řez. V červeně znázorněné hraně bude probíhat hledání protějšího bodu. Zobrazené nepřesnosti jsou pouze pro názornost. Podobrázek 4.20a znázorňuje nepovedený pokus o nalezení protějšího bodu, ale 4.20b již znázorňuje zdařilé nalezení tohoto bodu.

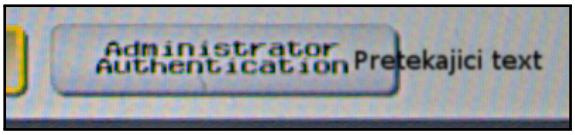
Detekce menu tlačítek vytvářela poměrně velké množství falešných nálezů. Proto nalezená tlačítka musí mj. splňovat podmínku na existenci textu uvnitř - ve středové části se musí nacházet hrany. Další podmínka omezuje jejich výskyt; například tlačítka horizontálního menu se mohou vyskytovat pouze v horní části obrazovky a tlačítka vertikálního menu se mohou vyskytovat pouze v levé části obrazovky.

4.5 Detekce přetékání textu

Tato část popisuje detekci přetékání podle tří různých přístupů. Tato kontrola je aplikována pouze na tlačítka, na kterých již v předchozí fázi nebylo detekováno přetékání, na základě jejich vzoru. Detekce přetékání je možná na základě nalezených textů, na základě hran vzniklých textem či na základě poškození tvaru kontury tlačítka.

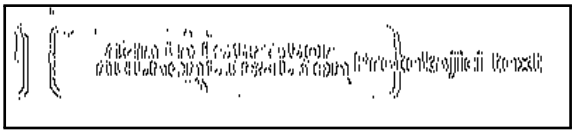
Detekce přetékání na základě hran vzniklých textem Základem pro získání hran způsobených textem je výstup Canny detektoru. Z tohoto výstupu jsou odfiltrovány horizontální a vertikální hrany. V případě kontroly správného zobrazení okrajů tlačítka je poté třeba následně kontrolovat, zda se v oblasti svislé hrany tlačítka nevyskytují hrany horizontální, a naopak. Tento způsob detekce nesprávného zobrazení je vhodný spíše pro kontrolu horizontálních hran tlačítka. Vertikální strana tlačítka bývá poměrně krátká a v případě

překrytí nevhodným textem je poměrně problematické odlišit hrany vzniklé textem od šumu pouze na základě vzniklých horizontálních hran. V textu se mohou vyskytovat pro tento účel nevhodné symboly, jako jsou mezery a znaky „I“ a podobně, které nevytvoří dostatečné množství horizontálních hran.

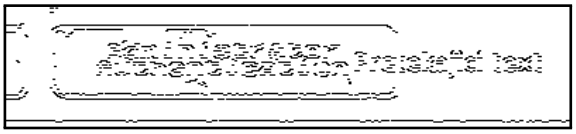


(a) Originální obrázek

(b) Hrany získané Canny detektorem hran



(c) Zanechány pouze vertikální hrany

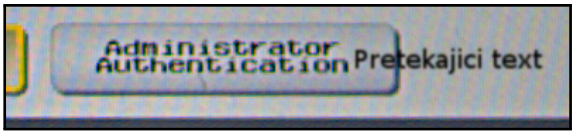


(d) Zanechány pouze horizontální hrany

Obrázek 4.21: Ukázka hran získaných pro detekci textu

Na obrázku 4.21 lze vidět znázornění horizontálních a vertikálních hran.

Detekce přetékání podle masky tlačítka Tento způsob je již teoreticky zahrnut v samotné detekci tlačítek, ve které detektor rozpozná tlačítko s přetékajícím textem dle tvaru kontury. Cílem tohoto detektoru je však primárně nalézt tlačítka. Detaily potřebné k detekci přetékání může tento detektor pro svou potřebu odfiltrovat. Detekce přetékání textu pomocí masky hledá v masce výřezy, které bývají způsobené textovými segmenty v oblasti tlačítka. Na obrázku 4.22c lze vidět kromě originální fotografie také černě znázorněnou masku. Na této masce jsou bílé oblasti, které mohou reprezentovat text. V případě, že se tyto oblasti dotýkají okrajů a splňují určitá kritéria, je detekováno přetékání textu.



(a) Originální fotografie

(b) Výstup segmentace

(c) Černě znázorněná maska tlačítka

(d) Maska se znázorněnou oblastí přetékání

Obrázek 4.22: Rozsegmentovaný obrázek a maska tlačítka

Detekce přetékání na základě OCR Jednou z dalších možných cest, jak detekovat přetékání textů, je hledání průniku oblastí nalezeného textu s hranami tlačítek. Tento způsob obvykle vyžaduje znalost jazyka textu užitého ve fotografii k tomu, aby bylo možno správně odfiltrovat falešně pozitivního nálezu. Kvalitu OCR výsledků negativně ovlivňuje nízké rozlišení a šum ve fotografiích. Může také docházet k neschopnosti detekovat text protínaný hranou tlačítka, což je právě nejdůležitější situace, kdy by bylo třeba text detekovat.

4.6 Sjednocení výsledků

V mezivýsledcích se mohou objevit rozpoznaná tlačítka, která se překrývají. Toto může nastat jednak v případě, kdy je stejné tlačítko detekováno více různými metodami, například jako klasické a menu tlačítko. Druhým případem mohou být například nesprávně detekovaná tlačítka. Falešně pozitivní nálezy mohou vzniknout v několika příkladech. První z těchto případů je označení ikony uvnitř tlačítka jako tlačítka. Druhý případ může nastat například tehdy, když je tlustý obrys tlačítka považován za samotné tlačítko. Dalším možným případem je situace, kdy je několik tlačítek ohraničeno obdélníkem, který je také rozpoznán jako tlačítko.

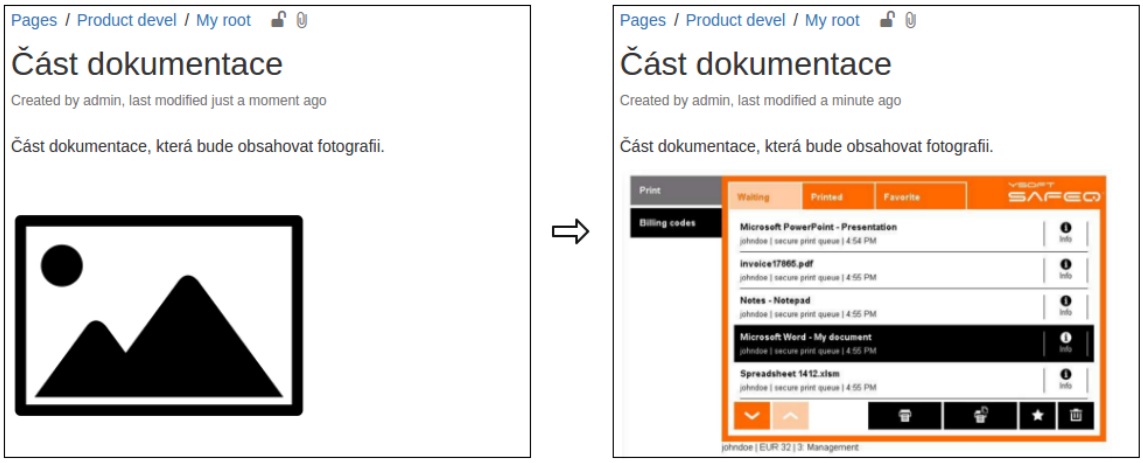
Prvním krokem této fáze je sjednocení shodných nálezů. Tento krok má za úkol detekovat množinu nálezů označující (přibližně) stejné místo a zvolit jeden nález, který zůstane. Volba závisí prakticky na třech faktorech. Pokud spolu kolidují dva nálezy, z nichž jeden reprezentuje poškozené tlačítko, je zachován tento nález, neboť nechceme přijít o detekovanou vadu. Pokud oba nálezy reprezentují vadu zobrazení, je vybrán ten, který má rozměry podobnější standardnímu tlačítku. Tyto rozměry jsou zadefinovány v parametrech pro detekci. V opačném případě je vybrán ten nález, který zabírá větší plochu.

V druhém kroku je prováděno odstraňování obecně překrývajících se nálezů. Pokud se detekovaná tlačítka jen částečně překrývají, jsou ponechána ve výsledcích. Pokud spolu kolidují pouze dvě tlačítka, je ponecháno to, které rozměry více připomíná standardní tlačítko. V případě, že s jedním nálezem koliduje více nálezů, je tento nález odstraněn, protože se pravděpodobně jedná o rámeček kolem několika tlačítek.

Každý krok je cyklicky prováděn tak dlouho, dokud dochází k odstraňování nálezů.

4.7 Nahrávání fotografií do dokumentace

Pro popis uživatelského rozhraní tiskárny je udržována online dokumentace v systému Confluence. Zde by měly být již připravené zástupné symboly (dále jen anglický výraz placeholder). Ty budou nahrazeny reálnými fotografiemi, jako je znázorněno například na 4.23. K dispozici je několik snímků znázorňujících stejnou scénu. Cílem je vybrat nejkvalitnější snímek a použít ho k nahrazení placeholderů v dokumentaci.



Obrázek 4.23: Nahrazení placeholderu reálným snímkem

Volba způsobu aktualizace fotografie v systému

Dokumentace je složena ze stromu jednotlivých stránek a vybraný obrázek by měl být nahrán na příslušná místa v určených podstromech. Tato místa je třeba označit vhodným placeholderem.

Volba placeholderu

Existuje několik způsobů, jak označit vhodné místo pro nahrání fotografie. Je důležité tato místa označit tak, aby je bylo možné vyhledat pomocí REST rozhraní systému a aby klientská aplikace nemusela stahovat a procházet jednotlivé stránky jednu po druhé a hledat zástupné symboly.

Textový řetězec je v textu snadno vyhledatelný, avšak při jeho použití není možné jednoduše uživatelsky definovat další parametry nahraného obrázku, jako například rámeček. Také je problém s aktualizací fotografie. V případě prvního nahrání fotografie musí dojít k přemazání zástupného symbolu. Ten poté nelze opětovně vyhledat, tedy ani nahradit.

Uživatelská makra se mohou jevit jako poměrně vhodná alternativa. Je možné obalit obrázek dokumentace do makro-elementu. V případě nahrání fotografie je možné upravit adresu fotografie uvnitř makra. Rozlišení mezi jednotlivými placeholdery je možné realizovat pomocí argumentu makra. Vyhledání potřebných maker v textu lze vykonat pomocí API rozhraní Confluence. Standardně však systém neumožňuje omezit vyhledávání dle hodnoty parametru makra.

Přímé nahrazení fotografie se jeví prakticky jako nejjednodušší a nejvhodnější volbou. Každá stránka obsahuje seznam všech příloh. Tyto přílohy jsou identifikovány jménem souboru. Systém udržuje několik revizí příloh a spolu s komentáři a s časovým razítkem se toto jeví za ideální řešení automatického nahrávání fotografií. Uživatel nahraje prázdnou fotografii obsahující například symbol chybějícího obrázku. Tato fotografie pak bude programem automaticky aktualizována.

Algoritmus výběru fotografie

Kvalitu fotografií lze hodnotit podle mnoha kritérií. V současné době je zvoleno pouze kritérium neostrosti. Jedná se o poměrně univerzální a momentálně dočasné řešení. Společnost Y Soft předpokládá obecné zlepšení kvality fotografií, které budou získávány. V tuto chvíli nelze určit jaké defekty budou fotografiích přetrvávat.

O určení míry rozostřenosti se stará třída `ImageFuziness`. Jedinou veřejnou metodou této třídy je metoda CountSharpness(filepath). Tato metoda načte zdrojový obraz v barevném modelu odstínů šedi. Z tohoto obrazu jsou v paměti vytvořeny dvě kopie. Na každou takovou kopii je poté aplikován Sobelův operátor - v jednom případě jeho vertikální a v druhém případě jeho horizontální varianta. V takových obrazech jsou poté vypočteny tloušťky čar a vypočten jejich průměr. Třída `BestImageSelector` se stará o paralelní zpracování jednotlivých obrazů a výběr nejvhodnějšího kandidáta.

(a) Průměrná šířka hran: 7.8px (b) Průměrná šířka hran: 16.14px

Obrázek 4.24: Vizualizace vypočtené rozostřenosti na základě průměrné šířky hran (zobrazeny pouze výřezy hodnoceného obrazu)

Realizace nahrávání fotografie

Pro komunikaci mezi C# a Confluence bylo použito REST rozhraní. Jako potřebný identifikátor cílové části dokumentace je použit URL odkaz na editaci kořenové stránky. Celý odkaz je použit také z toho důvodu, že je jej možné jednoduše získat z webového rozhraní systému Confluence, na rozdíl například od samotného číselného ID. Tento odkaz je aplikací poté rozparsován a je z něj získána jak základní URL adresa pro REST dotazy systému Confluence, tak číselné ID kořenové stránky.

Cílem aplikace je vyhledat všechny přílohy (Attachment) stránek, které mají požadované jméno souboru. Confluence však neumožňuje vyhledat přílohy pouze v daném dokumentačním podstromu. Vyhledávání probíhá ve dvou fázích: v první fázi jsou vyhledány všechny stránky obsahující jméno souboru. V druhé fázi jsou vyhledány všechny přílohy daného článku s potřebným jménem. Prohledávání příloh je důležité, neboť je třeba získat konkrétní ID dané přílohy. To je poté využito v dotazu pro nahrání obrazových dat.

4.8 Testování aplikace

Tato kapitola seznámí čtenáře s výsledky získanými automatickým vyhodnocením úspěšnosti a také názorně ukáže některé výsledky detekce.

Volba metriky

Vyhodnocení úspěšnosti detekce je poměrně komplikované, neboť vstupem pro hodnocení není pouze jedna hodnota, ale hned několik. Mezi tyto hodnoty lze řadit například procento úspěšně detekovaných či nedetekovaných tlačítek, počet falešných nálezů tlačítek, procento nesprávně detekovaného přetékání. Volba vhodné metriky pro hodnocení detekce tedy není jednoznačná. Pro výpočet úspěšnosti jsem se rozhodl popsat výsledky pouze pomocí tří proměnných, ze kterých je poté počítána procentuální úspěšnost 4.2. První parametr `valid` určuje počet správně detekovaných tlačítek - včetně přetékání. Druhý parametr `invalid` definuje počet nesprávně detekovaných tlačítek. V tomto případě se jedná o tlačítka, která nebyla vůbec nalezena, či u kterých bylo nesprávně detekováno přetékání. V případě kontroly přetékání je zohledněno, i přes kterou ze čtyř stran text přetéká. Třetím parametrem je `false`, který určuje počet falešných nálezů.

$$accuracy = 100 * \frac{valid}{valid + invalid + false} \tag{4.2}$$

Tato metrika tedy nemá za cíl porovnávat falešně pozitivní nálezy s falešně negativními nálezy. Jedná se prakticky o neporovnatelné veličiny a nelze jednoznačně říct, která chyba je závažnější. Na jednu stranu se může zdát, že falešně negativní nálezy jsou méně závažné. V praxi však i celkem malé procento falešně negativních nálezů může způsobit, že bude třeba manuální kontroly velkého množství fotografií a detektor by pozbýval smyslu. Také je problém s určováním těchto proměnných, neboť pro detekci falešně pozitivního/negativního nálezu přetékajícího textu přes hrany je třeba, aby bylo správně detekováno samotné tlačítko. Pokud by bylo hodnoceno tlačítko jako celek, mohly by nastat situace, kdy nelze tlačítko jednoznačně klasifikovat. Důvodem je fakt, že na některé hraně může dojít k falešně pozitivnímu nálezu a na jiné k falešně negativnímu.

Některé útvary v obraze mohou svými vlastnostmi připomínat tlačítka, ale o tlačítka se nejedná - například ohraničená záhlaví okna s textem. Nebo se také jedná o tlačítka, která není cílem detekovat - například čtvercová tlačítka s ikonou. Zahrnutí či nezahrnutí těchto útvarů ve výsledcích neovlivňuje počítanou úspěšnost. V případě, kdy detektor začne tyto útvary považovat za tlačítka, musí také správně rozpoznat, zda jsou validně zobrazeny.

Cílem testování bude zjistit průměrnou procentuální úspěšnost detekce tlačítek na snímcích z různých kategorií. Kromě toho také uvést celkovou úspěšnost, která bude popisovat procentuální množství naprosto správně rozpoznaných fotografií.

4.8.1 Realizace testů

Úspěšnost algoritmu byla zhodnocena na množině fotografií. Některé fotografie byly získány fotoaparátem, v jiném případě se jednalo o poměrně kvalitní obrázky, které nebyly získány fotoaparátem. Na obrázky bylo aplikováno zkreslení v podobě gaussovského rozostření a šumu, což mělo částečně simulovat snížení kvality způsobeného focením. To umožnilo zvýšit množství fotografií k testování a také zjistit vliv zkreslení na detekci.

Pro automatické vyhodnocení úspěšnosti bylo třeba získat anotace k jednotlivým fotografiím. Tyto informace byly vygenerovány samotnou detekční aplikací, poté pomocí aplikace B byly ručně zkontrolovány a upraveny. Před-generování těchto dat umožnilo ušetření manuální práce. Množina takto anotovaných fotografií byla následně rozšířena o zkreslené varianty těchto fotografií.

	Úroveň šumu (σ)				
Rozostřenost (**s**)		**-**	**4**	**8**	**16**
	-	**88%**	88%	77%	11%
	3	89%	89%	88%	87%
	5	86%	87%	86%	83%
	7	70%	72%	72%	75%

Tabulka 4.2: Úspěšnost detekce 21 obrázků z dokumentace pro různou úroveň zkreslení

Tabulky výše zobrazují úspěšnost detekce. Ta byla počítána jako průměrná úspěšnost dosažená pro jednotlivé fotografie. Vstupní data byla rozdělena do tří kategorií: reálné fotografie (4.4), obrázky z dokumentace (4.2) a umělé obrázky (4.3). Kategorii umělých obrázků lze zařadit do kategorie obrázků z dokumentace, neboť se jedná pouze o jejich modifikace.

	Úroveň šumu (σ)				
Rozostřenost (**s**)		**-**	**4**	**8**	**16**
	-	**99%**	98%	97%	18%
	3	99%	93%	97%	94%
	5	81%	80%	81%	84%
	7	60%	60%	61%	63%

Tabulka 4.3: Úspěšnost detekce 12 umělých obrázků pro různou úroveň zkreslení

	Úroveň šumu (σ)				
Rozostřenost (**s**)		**-**	**4**	**8**	**16**
	-	**82%**	79%	74%	8%
	3	78%	79%	76%	73%
	5	75%	74%	75%	76%
	7	70%	67%	69%	66%

Tabulka 4.4: Úspěšnost detekce 25 reálných fotografií pro různou úroveň přidaného zkreslení

Jedná se však o množinu podobných obrázků s vysokou mírou úspěšnosti detekce, což by ovlivnilo statistiku a neodráželo tolik reálný stav. Tabulky znázorňují úspěšnost v závislosti na přidaném zkreslení. Parametr **s** udává velikost jádra gaussovské filtrace a parametr σ udává směrodatnou odchylku použitého gaussovského šumu. V počátku tabulky lze vidět úspěšnost na nezkreslených fotografiích.

Pokud bychom hodnotily dle míry naprosto správně rozpoznaných fotografií, byla by celková úspěšnost napříč kategoriemi, bez umělých snímků, rovna **48%**.

Ukázkové detekce

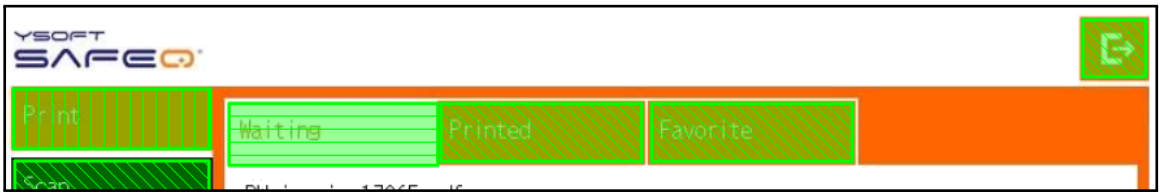
Pro otestování samotných detekcí přetékání byl zvolen primárně kvalitní referenční obraz, do kterého byly přidány různé druhy přetékání. Tento obraz netrpí obrazovými vadami, jako reálné fotografie.

Obrázek 4.25: Originální obraz

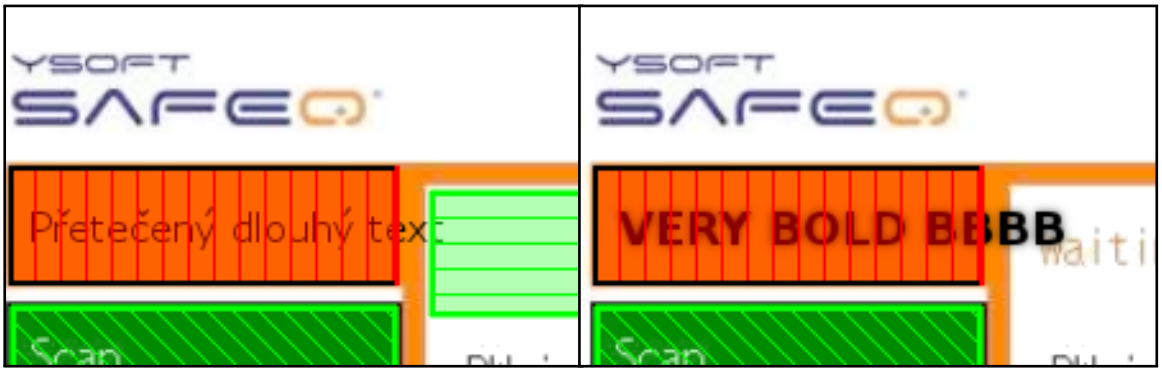
Obraz 4.25 znázorňuje podobu originálního obrazu. Jednotlivé příklady níže budou obsahovat jeho věcné modifikace.

Znázorněný výřez v 4.26 vyobrazuje detekovaná klasická i menu tlačítka. Rozpoznaný druh tlačítka lze určit dle typu šrafování. Pro běžné použití není typ tlačítka podstatný. Tato informace je vhodná spíše pro ladění. Jak již bylo zmíněno v textu výše, menu tlačítka je třeba detekovat odlišným způsobem díky absenci jedné ze čtyř jejich hran.

Na obrázcích 4.27 lze vidět poškození vertikálního tlačítka menu dvěma typy textu. Tučný text obecně způsobuje změnu tvaru kontury segmentu tlačítka. V případě vertikálního tlačítka tvar pravé strany však nehraje roli při jeho rozpoznávání. Naopak zde vidíme,



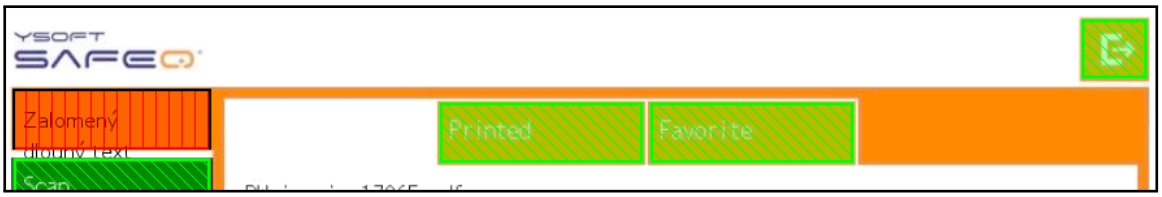
Obrázek 4.26: Originální obrázek s výsledky detekce



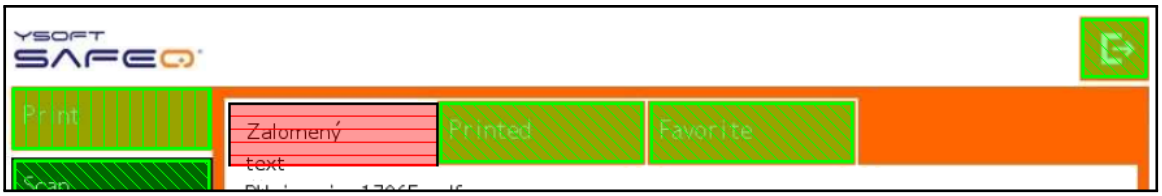
(a) Poškození tenkým textem (b) Poškození tlustým textem

Obrázek 4.27: Poškození pravé strany tlačítka vertikálního menu

že poškozené tlačítko horizontálního menu nebylo detekováno. To je způsobeno tím, že není definován vzor pro detekci takto poškozených tlačítek - jinými slovy lze říct, že aplikace s tímto typem přetékání nepočítá.



(a) Zalomený text vertikálního tlačítka menu

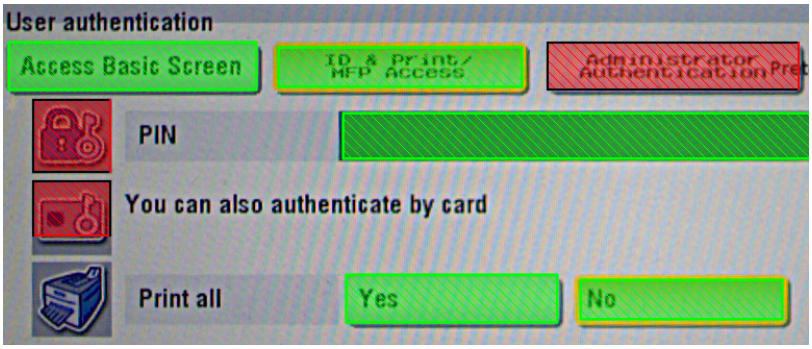


(b) Zalomený text horizontálního tlačítka menu

Obrázek 4.28: Zalomení textu

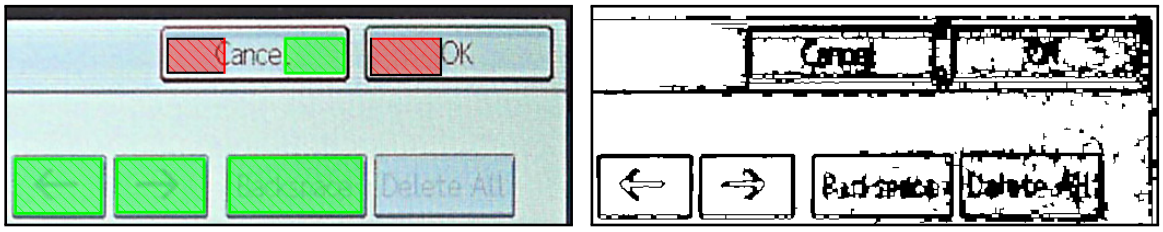
Obrázky 4.28 reprezentují problém zalomeného textu. Tento text by neměl ovlivňovat detekované kontury tlačítek a je tedy detekován až následnou detekcí přetékání. V případě 4.28b lze vidět, že i když text nepřetéká přes žádnou viditelnou hranu, je detekován jako přetékající.

V případě použití detektoru na reálných fotografiích se výsledky často liší dle typu fotografie. Kvalita výsledků je poměrně silně závislá na typu grafiky na fotografii.



Obrázek 4.29: Ukázka s reálnou fotografií č.1

Příklad 4.29 ukazuje, že detektor se do jisté míry zvládne vypořádat i s určitým množstvím šumu. Problematická jsou však tlačítka ikon. Ta by šla v případě znalosti obrazu snadno vyfiltrovat. Popisky `PIN` a `Print all` by bylo vhodné také považovat za tlačítka, bohužel je jejich barva příliš podobná barvě pozadí a v množství šumu je nelze rozpoznat.



(a) Výřez špatně detekovaných tlačítek (b) Výstup canny detektoru + uzavření

Obrázek 4.30: Reálná fotografie č.2

Při pohledu na 4.30a není na první pohled patrné, proč detekce selhala. Pohlédneme-li na hranový obraz 4.30b použitý k segmentaci, vidíme velké množství hran navíc. Toto by bylo řešitelné v případě, že by byly parametry detektoru upraveny pro tento typ fotografie.

Zhodnocení

Na připravených obrazech je detektor schopen rozeznat různé druhy tlačítek stejně jako různé typy jejich přetékání. Úspěšnost detekce se v některých případech může velmi lišit na základě zvoleného testovacího vzorku. Některé typy stylů tlačítek mohou být se současně zvolenými parametry detektoru spolehlivěji rozpoznatelné než jiné. Zvolený způsob detekce tlačítek a přetékajícího textu je teoreticky univerzální pro většinu tlačítek připomínajících obdélník. Hlavními dvěma faktory ovlivňující výsledky je tedy nastavení parametrů detektoru a seznam definovaných vzorů pro detekci tlačítek s přetékajícím textem. V případě, že detektor musí obecně pracovat i s fotografiemi nízké kvality, lze očekávat horší výsledky i na fotografiích kvalitních. Detektor musí provádět určitý způsob filtrace a ignorovat potenciální šum. To může způsobit, že i na kvalitních fotografiích nebude detekovaná pozice tlačítek přesná a nevýrazně přetékající text bude ignorován, neboť bude považován za šum. Detekce případů, na kterých nevýrazný text přetéká přes okraj, je obecně univerzální, neboť neovlivňuje konturu tlačítka (s určitým vyfiltrováním). V případě přetékajícího tučného textu, který konturu ovlivní, je však situace jiná. Je třeba nadefinovat vzory pro detekci takovýchto úkazů, jako například tlačítko horizontálního menu s přetékajícím textem vlevo,

s kterým v současnou dobu detektor nepočítá. V případě, že by standardně akceptoval tlačítka s různými deformacemi kontury, dalo by se očekávat větší množství falešně pozitivních nálezů. Problematickým případem pro detekci může být text bez mezer, který protíná celou šířku tlačítka a prakticky dělí tlačítko na dvě poloviny.

Kapitola 5

Závěr

Cílem práce bylo naimplementovat postup pro kontrolu zobrazení textů ve formulářích a navrhnout vhodný způsob výběru fotografie k dokumentačním účelům. Tento cíl byl splněn.

Pro možnost realizace práce byla nastudována literatura na téma práce s obrazem a textem. Jedná se například o princip Houghovy transformace použitelné k detekci přímek či obdélníků, ale také obecně využitelné segmentační metody či možný postup rozpoznání textu v obraze. Bylo navrženo několik postupů detekce tlačítek. Jeden z nich byl kompletně implementován včetně kontroly přetékání textů. K užití fotografií v dokumentaci byl implementován automatický výběr fotografie s možností nahrání do dokumentačního nástroje. Výsledky práce byly znázorněny na příkladech a také v tabulkách popisujících úspěšnost detekce.

Přestože se práce zabývá kontrolou zobrazení textů, jejich samotný obsah díky nespolehlivosti získávaných výsledků rozpoznáván není. Byly však užity postupy, které přetékající text detekují s větší spolehlivostí. Nevýhodou řešení bez kontroly obsahu je nemožnost detekce oříznutých textů, u kterých není vizuálně patrné, že jsou oříznuté - tedy v případech, kdy dojde k oříznutí v oblasti mezery.

Práce byla laděna na dostupných obrazech společnosti Y Soft. U některých obrazů se jednalo o fotografie se zkreslením způsobeném focením, některé snímky byly získány z dokumentace, jiné byly ručně upraveny za účelem přidání přetékajícího textu. Obecně ne na všech snímcích, na kterých byla detekce testována, je třeba ji provádět. Na druhou stranu také platí, že ne na všech snímcích, na kterých má být detektor používán, byl testován. Je možné, že pro jiné snímky, či vylepšení výsledků na konkrétních snímcích, by bylo třeba upravit parametry detektoru.

Výsledná aplikace je použitelná pro detekci tlačítek a také pro detekci přetékání textu. Možné užití této aplikace si lze představit i s klasickými formuláři vykreslovanými například webovým prohlížečem. Dalším možným řešením je využití detekovaných tlačítek jako základu pro automatickou tvorbu navigační mapy tiskáren.

Jako možnost pokračování práce bych určitě zvážil specializaci detektoru pro určité typy fotografií. Toto rozšíření by však po uživateli vyžadovalo zařazení vstupů do určitých kategorií. Detektor by buď mohl zvolit pouze jiné parametry detekce, či k detekci přistupovat naprosto specializovaným způsobem. Máme-li informace o přibližném rozložení stránky a očekáváme-li například tlačítka pouze v některých oblastech, lze samotný postup detekce výrazně zjednodušit a prakticky se vyhnout falešným nálezům. Zlepšila by se tak spolehlivost detektoru a snížil by se čas nutný na dobu manuální kontroly falešně nalezených tlačítek.

Literatura

[1] ABBYVIDEOS: ABBYY: The Benefits of OCR Technology. In *Youtube*, 2016.
URL https://www.youtube.com/watch?v=WQ1MUHmEjLA

[2] Anuradha, G.: *An investigation into Telugu font and character recognition.* Dizertační práce, University of Hyderabad, Hajdarábád, 2009.
URL http://shodhganga.inflibnet.ac.in/handle/10603/4166

[3] Bassil, Y.; Alwani, M.: OCR Post-Processing Error Correction Algorithm using Google Online Spelling Suggestion. *ArXiv e-prints*, Duben 2012, ISSN 2079-8407.
URL https://arxiv.org/pdf/1204.0191v1.pdf

[4] Bigas, M.; Cabruja, E.; Forest, J.; aj.: Review of CMOS image sensors. *Microelectronics Journal*, ročník 37, č. 5, 2006: s. 433–451, ISSN 00262692, doi:10.1016/j.mejo.2005.07.002.
URL http://www.sciencedirect.com.ezproxy.lib.vutbr.cz/science/article/pii/S0026269205002764

[5] Borovikov, E.: A survey of modern optical character recognition techniques. *ArXiv e-prints*, Prosinec 2014.
URL https://arxiv.org/pdf/1412.4183v1.pdf

[6] Burger, W.; Burge, M.: *Principles of digital image processing.* London: Springer, c2009, ISBN 978-1-84800-191-6.

[7] Comaniciu, D.; Meer, P.: Mean shift. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, ročník 24, č. 5, 200205: s. 603–619, ISSN 01628828, doi:10.1109/34.1000236.
URL https://www.cse.unr.edu/~bebis/CS773C/ObjectRecognition/Papers/Comaniciu02.pdf

[8] Curtin, D.: *The textbook of digital photography.* Marblehead, Mass: ShortCourses.com, druhé vydání, 2007, ISBN 19-288-7375-8.

[9] Doubek, P.: *Mean-Shift segmentace.* Praha: CMP FEL ČVUT, 2007.
URL http://cmp.felk.cvut.cz/cmp/courses/ZS1/Cviceni/cv4/meanshift.pdf

[10] Øivind Due Trier; Jain, A. K.; Taxt, T.: Feature extraction methods for character recognition-A survey. *Pattern Recognition*, ročník 29, č. 4, 1996: s. 641–662, ISSN 00313203, doi:10.1016/0031-3203(95)00118-2.
URL https://www.cs.sfu.ca/CourseCentral/414/li/material/refs/OCR-survey-96.pdf

[11] Eikvil, L.: *Optical Character Recognition*. Oslo: Norsk Regnesentral, 1993.
URL http://www.nr.no/~eikvil/OCR.pdf

[12] Fielding, R.: *Architectural Styles and the Design of Network-based Software Architectures*. Dizertační práce, Department of Computer Science, 2000.
URL
http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[13] Gonzalez, R. C.; Woods, R. E.: *Digital image processing*. Upper Saddle River, N.J.: Pearson Prentice Hall, třetí vydání, c2008, ISBN 978-0-13-505267-9.

[14] Janvalkar, S.; Manjrekar, P.: Text Recognition from an Image. *International Journal of Engineering Research and Applications*, ročník 4, č. 4, 2014: s. 149–151, ISSN 22489622.
URL http://ijera.com/papers/Vol4_issue4/Version%205/Z04405149151.pdf

[15] Jung, C.; Schramm, R.: Rectangle detection based on a windowed Hough transform. In *Computer Graphics and Image Processing, 2004. Proceedings. 17th Brazilian Symposium on*, USA: IEEE, 2004, ISBN 0769522270, ISSN 15301834, s. 113–120, doi:10.1109/SIBGRA.2004.1352951.

[16] Kybic, J.: *Mean Shift Segmentation*. ČVUT FELK, 2007.
URL
http://cmp.felk.cvut.cz/cmp/courses/33DZOzima2007/slidy/meanShiftSeg.pdf

[17] Marziliano, P.; Dufaux, F.; Winkler, S.; aj.: A no-reference perceptual blur metric. In *Image Processing. 2002. Proceedings. 2002 International Conference on*, ročník 3, USA: IEEE, 2002, ISBN 0780376226, ISSN 15224880, s. III–III, doi:10.1109/ICIP.2002.1038902.

[18] Parker, J. R.: *Algorithms for image processing and computer vision*. New York: Wiley Computer Pub., 1997, ISBN 04-711-4056-2.

[19] Qidwai, U.; Chen, C.: *Digital Image Processing: An Algorithmic Approach with MATLAB*. Chapman & Hall/CRC Textbooks in Computing, CRC Press, 2009, ISBN 9781420079517.
URL https://books.google.cz/books?id=PwXLBQAAQBAJ

[20] Ren, Y.-Z.; Domeniconi, C.; Zhang, G.; aj.: A Weighted Adaptive Mean Shift Clustering Algorithm. In *SDM*, SIAM, 2014, s. 794–802.
URL http://epubs.siam.org/doi/pdf/10.1137/1.9781611973440.91

[21] R.Yogamangalam; B.Karthikeyan: Segmentation Techniques Comparison in Image Processing. *International Journal of Engineering and Technology*, ročník 5, č. 1, 2013: s. 307–313, ISSN 09754024.

[22] Shapiro, L. G.; Stockman, G. C.: *Computer vision*. Upper Saddle River, NJ: Prentice Hall, 2001, ISBN 01-303-0796-3.

[23] Slobojan, R.: *InfoQ Explores: REST*. InfoQ, první vydání, 2010.
URL https://www.infoq.com/minibooks/emag-03-2010-rest

[24] Wang, Z.; Simoncelli, E. P.: Reduced-reference image quality assessment using a wavelet-domain natural image statistic model. In *Electronic Imaging 2005*, International Society for Optics and Photonics, 2005, s. 149–159.

[25] Wikipedia: CuneiForm (software) — Wikipedia, The Free Encyclopedia. 2016, [Online; accessed 15-December-2016].
URL https://en.wikipedia.org/w/index.php?title=CuneiForm_(software)&oldid=754913807

[26] Wikipedia: C Sharp (programming language) — Wikipedia, The Free Encyclopedia. 2017, [Online; accessed 15-May-2017].
URL https://en.wikipedia.org/w/index.php?title=C_Sharp_(programming_language)&oldid=778436464

[27] Wikipedia: Charge-coupled device — Wikipedia, The Free Encyclopedia. 2017, [Online; accessed 15-May-2017].
URL https://en.wikipedia.org/w/index.php?title=Charge-coupled_device&oldid=777650133

[28] Xu, S.; Jiang, S.; Min, W.: No-reference/blind image quality assessment. *IETE Technical Review*, 2016: s. 1–23, ISSN 02564602, doi:10.1080/02564602.2016.1151385.

[29] Šikudová, E.: *Počítačové videnie*. Praha: Wikina, první vydání, [2014], ISBN 978-80-87925-06-5.

Přílohy

Příloha A

Obsah CD

- **/examples** - ukázky fotografií
  - **/photos** - nerozpoznané fotografie + anotace
  - **/detected** - detekovaná tlačítka včetně detekce přetékání
  - **/results.csv** - zhodnocení úspěšnosti pro jednotlivé fotografie
- **/ButtonsProcessor** - zdrojové soubory
  - **/ButtonsSelector(.Desktop)** - aplikace pro definici anotací
  - **/ButtonsChecker** - aplikace pro rozpoznání a kontrolu tlačítek
  - **/ConfluenceUploader** - aplikace pro nahrávání do dokumentace
  - **/ReferenceDataGenerator** - aplikace pro generování šumových fotografií
  - **/Tests** - jednotkové testy
- **/bin** - zkompilované aplikace
  - **/*.exe** - Zkompilované aplikace z výše uvedených zdrojových kódů
- **/doc** - zdrojové soubory k tomuto dokumentu
- **/zprava.pdf** - tento dokument

Příloha B

Praktické použití aplikací

Aplikace je napsána v jazyce C#. Kompilace byla testována na 64 bitové verzi Windows s Microsoft Visual Studiem 2015. Po otevření solution z hlavního adresáře zdrojových kódů lze jednotlivé projekty zkompilovat. Pro samotné spuštění aplikací stačí však zvolit zkompilované varianty dostupné na odevzdaném disku.

Detekce tlačítek

Hlavní činností aplikace je detekce přetékání textů v tlačítkách. Parametry programu zadaných z příkazové řádky lze ovlivnit, jak s výsledky aplikace naloží.

Výsledky je možné uložit ve formě referenčních dat pro hodnocení úspěšnosti algoritmu pomocí parametru –**reference**. Jelikož ruční vyplňování referenčních dat je pro takové výsledky poměrně zdlouhavé, je možné je vygenerovat samotným programem a poté pouze manuálně opravit. Vedle každé vstupní fotografie bude vygenerován csv soubor s rozpoznanými tlačítky.

Další možností je uložení všech rozpoznaných fotografií do cílového adresáře pomocí parametru –**output**. Vygenerované soubory budou obsahovat původní fotografie spolu se zobrazenými rozpoznanými tlačítky. Tlačítka jsou tří typů - klasické tlačítko, tlačítko horizontálního menu, tlačítko vertikálního menu. Typ tlačítka lze rozpoznat podle stylu šrafování. Tlačítko horizontálního menu má horizontální šrafování, tlačítko vertikálního menu má šrafování vertikální a klasické tlačítko má šrafování diagonální. Rozpoznaná tlačítka, u kterých nebylo rozpoznáno přetékání, jsou vyobrazeny zeleně, v opačném případě mají červenou barvu. Jejich 4 hrany jsou černé barvy kromě těch, u kterých bylo detekováno přetékání.

Pomocí parametru –**accuracy** lze vyhodnotit úspěšnost detekce na základě referenčních souborů nacházejících se u vstupních dat.

Parametr –**check** slouží k uložení kompaktních výsledků, které obsahují pouze fotografie, v kterých byly rozpoznány vady.

Na ukázce níže je aplikace použita na zpracování všech fotografií ve složce `input` a výsledky jsou uloženy do složky `output`. Nalezené chyby v zobrazení jsou uloženy do složky `check`.

Kód B.1: Ukázka použití.

```
ButtonsChecker.exe -i input -o output -c check
Input directory:  input (2 images found)
Check results:  check
```

```
Images output directory:  output
Write reference data: no
Processing started
Processed(input/doc_s11.png): 1./2 images. Elapsed
   00:00:03.6179133.
Processed(input/doc_s10.png): 2./2 images. Elapsed
   00:00:03.9532864.
Total time: 00:00:03.9607149
Invalid photos: 2/2
Save: check/doc_s10.png
Save: check/doc_s11.png
Saving done
Save: output/doc_s10.png
Save: output/doc_s11.png
Saving done
```

Kód B.2: Nápověda k aplikaci.

```
Help:
-i, --input=VALUE          input photos directory tree with
   PNG files
-a, --accuracy=VALUE       accuracy output file
-o, --output=VALUE         images output dir
-c, --check=VALUE          define output directory for
   validation results
-r, --reference            write reference files
-h, --help                 show this message and exit
```

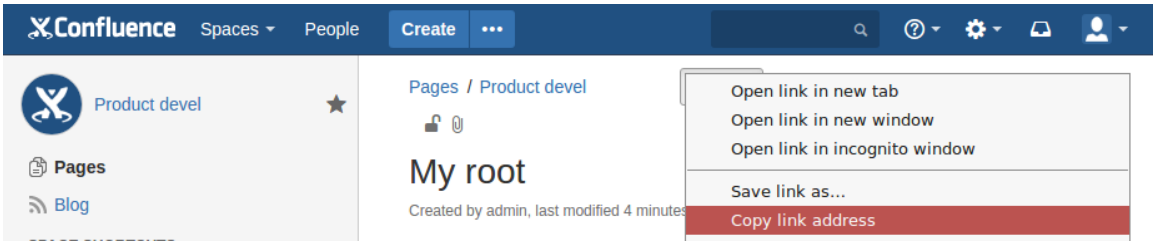
Nahrávání do dokumentace

Jedná se o konzolovou aplikaci. Chod aplikace lze upravit pomocí parametrů definovanými z příkazové řádky.

Kód B.3: Nápověda aplikace.

```
Usage:
<program> <arguments> images...
Arguments:
-u, --upload=VALUE         upload photos into confluence with
   tree link
-a, --placeholder=VALUE    placeholder filename
-s, --save                 save creditials into file
-l, --login=VALUE          login
-m, --comment=VALUE        comment
-c, --credentials=VALUE    creditials file
-p, --password=VALUE       password (not safe on multi-user
   system)
-i, --interactive          insert password interactively
-h, --help                 show help
```

V ukázce níže lze vidět reálné použití aplikace pro nahrání jedné z fotografií img1.png či img2.png do dokumentačního stromu uloženého na dané adrese. Jako uživatelské jméno je použito `admin` a heslo je zadáno interaktivně. Aplikace zvolila k nahrání soubor img1.png,



Obrázek B.1: Získání odkazu na strom dokumentační strom

který nahrála do dokumentace pod jménem souboru 3_051.png. V první řadě je třeba získat odkaz na část stromu v dokumentaci, v kterém chceme provést nahrazení placeholderů B.1. To lze provést tak, že zkopírujeme odkaz tlačítka pro editaci kořenové stránky.

Kód B.4: Ukázka použití.

```
ConfluenceUploader.exe -u
   http://pb:8090/pages/editpage.action?pageId=65591 -l admin
   -i -a 3_051.png img1.png img2.png
Type your password:
*****
Fuziness:       Filename:
6,68            img1.png
23,2            img2.png

Used creditials:  admin:*****
Target filename:  3_051.png

Found 1 relevant pages

Page: #65591 "My root"
Uploading...
Uploading done.
```

Generování zkreslených fotografií

Jedná se o jednoduchou aplikaci určenou pouze pro pomocné použití. Ukázka níže znázorňuje generování šumových fotografií. Originální fotografie jsou čteny ze složky `input` a jejich zdeformované varianty uloženy ve složce `output`.

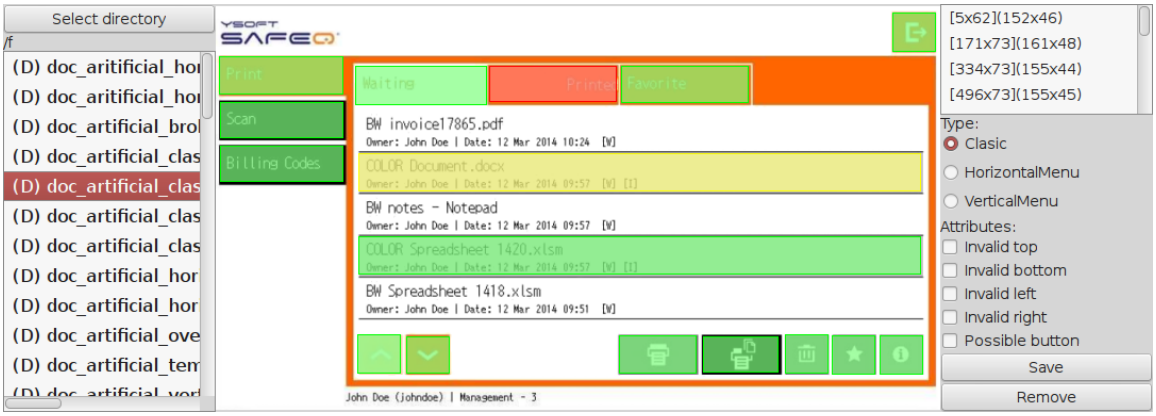
Kód B.5: Vygenerování fotografií s šumem.

```
ReferenceDataGenerator.exe input output

Original:       input\img1.png
Saving to output\img1_0_0.png
Saving to output\img1_0_4.png
Saving to output\img1_0_8.png
Saving to output\img1_0_16.png
...
```

Vyplňování anotací

Za účelem vyplnění referenčních dat, které je možné použít k hodnocení úspěšnosti algoritmu, byla napsána jednoduchá grafická aplikace. V této aplikaci je možné vybrat jednotlivá tlačítka a také jejich očekávané vady. Původní data pro tuto aplikaci jsou získána jako výsledek detektoru tlačítek. Získaná data je možno upravit v grafické aplikaci a následně použít k vyhodnocení kvality detektoru. Jako grafický toolkit byl zvolen Eto.Forms. Tento toolkit podporuje několik enginů, jako jsou například Gtk2, Gtk3, WinForms, WPF a další.



Obrázek B.2: Screenshot aplikace

Aplikace znázorňuje anotovaná tlačítka třemi barvami: zelená - tlačítka bez vad, červená - tlačítka s přetékajícím textem, žlutá - aktuální tlačítko. U každého tlačítka je zvolen typ (horizontální/vertikální menu či klasické tlačítko) a také charakter jeho přetékání.