VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

AUTOMATICKÁ ÚPRAVA OFOCENÝCH DOKUMENTŮ

AUTOMATIC ADJUSTMENT OF PHOTOGRAPHED DOCUMENTS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **LUKÁŠ KUCHTA**
AUTHOR

VEDOUCÍ PRÁCE — **JAROSLAV ROZMAN, Ing., Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Kuchta Lukáš**
Program: Informační technologie
Název: **Automatická úprava ofocených dokumentů**
Automatic Adjustment of Photographed Documents
Kategorie: Zpracování obrazu

Zadání:

1. Nastudujte problematiku geometrických operací s obrázkem (rotace, posunutí), dále nastudujte metody detekce stránek a textu na stránkách a metody úpravy obrázků.
2. Navrhněte knihovnu, která automaticky natočí ofocenou knihu rovnoběžně s rovinou fotoaparátu, dále bude umět ve fotografii nalézt okraje stránek a text na stránkách. Knihu rozdělí ve hřbetu na dvě stránky a použije na ně metody pro jasové úpravy obrázků. Pokuste se také vyrovnat ohnuté řádky blízko hřbetu knihy.
3. Navrženou knihovnu implementujte a vytvořte ukázkový program, který ji bude používat a bude uživatelsky co nejpříjemnější.
4. Vytvořený program a knihovnu otestujte.

Literatura:

- Yi Ma, An invitation to 3-D Vision: from images to geometric models, New York, Springer, 2006
- Richard Hartley, Andrew Zisserman, Multiple view geometry in computer vision, Cambridge, Cambridge University Press, 2003

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání.

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

Vedoucí práce: **Rozman Jaroslav, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 11. listopadu 2020

Abstrakt

Táto práca sa zaoberá automatickou úpravou vyfotených dokumentov, konkrétne vytvorením knižnice, ktorá sa postará o orezanie, zarovnanie, transformáciu a jasovú úpravu vyfotených dokumentov. Cieľom je využitie tejto knižnice na vytvorenie programu, do ktorého sa nahrá fotografia dokumentu a ten vráti upravenú verziu dokumentu. Na realizáciu knižnice sú využité funkcie knižnice OpenCV pre programovací jazyk Python, konkrétne funkcie na vyhľadanie okrajov strán dokumentu, ich orezanie, zarovnanie a úpravu jasu. Na vytvorenie grafického užívateľského rozhrania výsledného programu sa používa knižnica PyQt. Cieľ práce bol splnený, bol vytvorený program s príjemným užívateľským rozhraním na automatickú úpravu vyfotených dokumentov. Prínosom tejto práce je možnosť využitia vytvoreného programu na detekciu dokumentov v snímke a ich úpravu.

Abstract

This work deals with the automatic editing of photographed documents, specifically the creation of a library that will take care of cropping, alignment, transformation and brightness editing of photographed documents. The goal is to use this library to create a program into which a photo of the document is uploaded and which returns a modified version of the document. The functions of the OpenCV library for the Python programming language are used for the implementation of the library, specifically the functions for finding the edges of the document pages, their cropping, alignment and brightness adjustment. The PyQt library is used to create the graphical user interface of the resulting program. The goal was fulfilled, a program with a pleasant user interface for automatic editing of photographed documents was created. The benefit of this work is the possibility of using the created program to detect documents in the image and edit them.

Kľúčové slová

spracovanie obrazu, automatická úprava dokumentov, detekcia dokumentov, úprava jasu dokumentov, OpenCV, PyQt

Keywords

image processing, automatic document editing, document detection, document brightness adjustment, OpenCV, PyQt

Citácia

KUCHTA, Lukáš. *Automatická úprava ofocených dokumentů*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Jaroslav Rozman, Ing., Ph.D.

Automatická úprava ofocených dokumentů

Prehlásenie

Prehlasujem, že túto bakalársku prácu som vypracoval samostatne pod vedením pána Jaroslava Rozmana, Ing., Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Lukáš Kuchta
10. mája 2021

Poďakovanie

Rád by som využil túto príležitosť na poďakovanie vedúcemu práce pánovi Jaroslavovi Rozmanovi, Ing., Ph.D. za pomoc pri riešení práce a pri hľadaní zdrojov, taktiež by som sa rád poďakoval autorom publikácií, z ktorých som čerpal a v neposlednom rade aj autorom knižnice OpenCV a PyQt.

Obsah

Kapitola 1

Úvod

Určite ste sa už niekedy ocitli v situácii, kedy ste potrebovali vyfotiť nejaký dokument v papierovej forme a uložiť si ho v mobilnom telefóne alebo na počítači a ďalej s ním pracovať. Alebo ste od niekoho potrebovali vyfotený dokument a dotyčná osoba Vám poslala nekvalitnú snímku, pretože papier, ktorý bol fotený, bol pokrčený, alebo bol vyfotený v zlom svetle, prípadne s nejakými odleskami. To isté sa môže stať aj Vám. Som si istý, že najviac času pri fotení dokumentu venujete tomu, aby bol snímaný dokument pekne zarovnaný, aby bol zaostrený, mal dobré svetlo, žiadne odlesky. Predstavte si situáciu, že Vám dá niekto za úlohu zdigitalizovať nejakú knihu. Nastaviť knihu tak, aby bola v dobrom svetle a hlavne, aby strany neboli ohnuté. To by zabralo príliš veľa času.

Presne týmto sa zaoberá táto práca. Jej hlavným cieľom je minimalizovať námahu pri fotení dokumentov. Postarať sa o to, že Vám stačí nahrať fotografiu do aplikácie a tá automaticky oreže snímku, vycentruje ju, vyrovná ohnuté strany, a upraví jas pre lepšiu čitateľnosť.

Asi si myslíte, že už existuje množstvo takýchto aplikácií. To je z časti pravda, ale tieto aplikácie majú hneď niekoľko problémov. Väčšina aplikácií je určená pre mobilné telefóny, ale my potrebujeme aplikáciu, ktorá bude určená pre počítače. Ďalší problém týchto aplikácií je ten, že sa zameriavajú na skenovanie len jedného listu papiera, zatiaľ čo jedným z hlavných cieľov našej aplikácie je pracovať s jedným listom ale taktiež aj s dvoma listami v prípade, že potrebujeme spracovať snímku, ktorá obsahuje otvorenú knihu, teda dve spoločne spojené strany. Taktiež veľká časť týchto aplikácií nepracuje úplne automaticky, je potrebné buď označiť okraje strán, prípadne si upraviť jas samostatne. Nehovoriac o tom, že množstvo mobilných aplikácií, ale hlavne desktopových aplikácií je komerčných. Niektorým z nich sa budeme venovať v druhej kapitole.

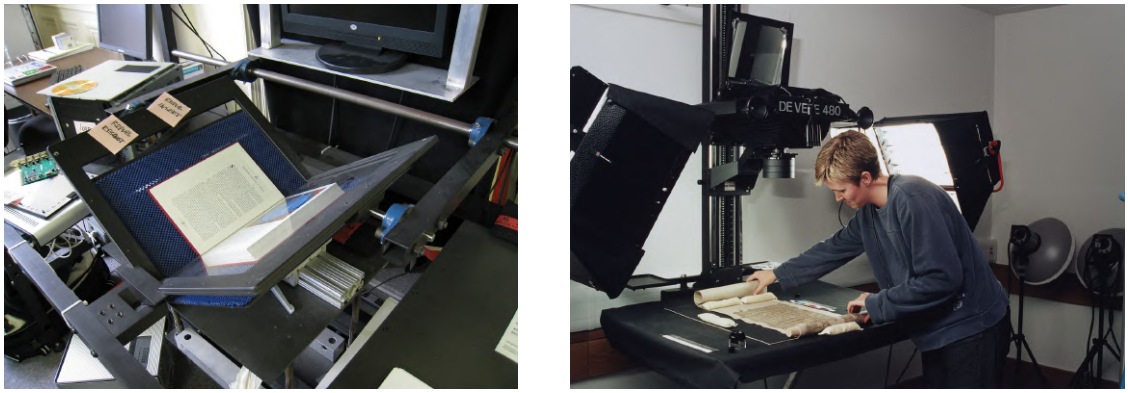
V tretej kapitole sa budeme venovať konkrétnym operáciám, ktoré je potrebné vykonať pri automatickej úprave dokumentov. Najskôr začneme stručným úvodom do spracovania obrazu a jeho digitalizáciou. Následne si rozoberieme techniky používané pre detekciu dokumentu v snímke, jeho orezanie, transformačné úpravy a úpravy jasu.

V štvrtej kapitole sa zoznámime s návrhom riešenia, použitím jazyka Python, implementáciou knižnice s použitím knižnice OpenCV a návrhu užívateľského rozhrania pre aplikáciu. V ďalšej kapitole nasleduje implementácia spomínaných častí. Na záver sa budeme zaoberať testovaním vytvorenej knižnice a vyhodnotením jej úspešnosti.

Kapitola 2

Súčasný stav

Na naskenovanie jedného listu papiera existuje niekoľko aplikácií, prevažne určených pre mobilné telefóny. Skenovanie kníh však väčšinou prebieha tak, že sa otvorená kniha položí na sklenenú platňu listami dole a pod ňou sa pohybuje snímač, alebo je kniha položená listami hore do tvaru V, aby sa minimalizovalo zakrivenie strán pri väzbe knihy a strany sa snímajú dvoma snímačmi umiestnenými oproti stranám. Čo sa týka softvérového riešenia, snímky sú upravované pomocou knižníc na úpravu obrázkov, ako napríklad OpenCV, alebo rozpoznávaním textu (OCR - optical character recognition). Moderný OCR softvér využíva neurónové siete, ktoré detekujú celé riadky textu.



(a) Zdroj[1] (b) Zdroj[2]

Obr. 2.1: **Súčasný stav digitalizácie dokumentov**

V tejto kapitole sa zameriame na preskúmanie aplikácií, ktoré sa zaoberajú rovnakým problémom, ako aplikácia vytvorená v tejto práci. Samozrejme, že už existujú riešenia tohto problému, ale je otázne, aké sú kvalitné a či by spĺňali ciele tejto práce. V ďalšej časti budú preskúmané desktop aplikácie, ale taktiež sa v krátkosti pozrieme aj na aplikácie určené pre mobilné telefóny.

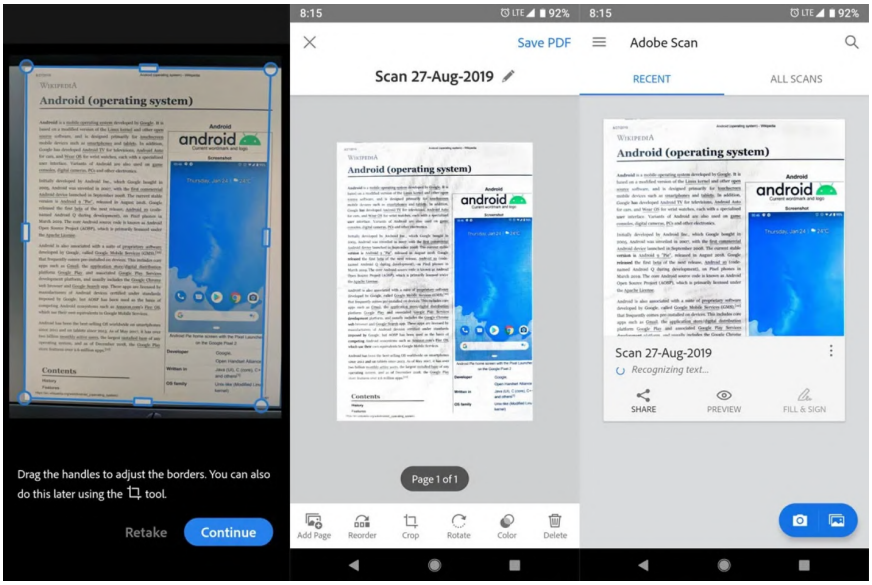
[1] https://upload.wikimedia.org/wikipedia/commons/6/65/Internet_Archive_book_scanner_1.jpg
[2] https://upload.wikimedia.org/wikipedia/commons/0/01/Dunhuang_manuscript_digitisation.jpg

2.1 Mobilné aplikácie

Tieto aplikácie sú vhodné pre ľudí, ktorí občas potrebujú naskenovať nejaký dokument. Sú rýchle, majú celkom dobré výsledky, ale nedokážu naskenovať otvorenú knihu a upraviť jej zahnuté strany. [7]

Adobe Scan

Bezplatná (v základnej verzii) mobilná aplikácia pre nenáročných užívateľov. Spočiatku sa môže zdať, že poskytuje málo možností, ale vďaka tomu je vhodná pre nenáročných užívateľov, ktorí potrebujú raz za čas naskenovať nejaký dokument. Má veľmi dobré výsledky pri rozpoznávaní textu a taktiež Vaše súbory ukladá na Adobe cloud (táto možnosť sa nedá vypnúť, nie je to teda vhodné pre scanovanie dokumentov s citlivým obsahom). Používa sa jednoducho, po spustení aplikácie je potrebné sa prihlásiť. Následne pomocou kamery naskenujete dokument, aplikácia by ho mala automaticky rozpoznať, je možné upraviť okraje dokumentu a pohrať sa s farbou. Dokumenty sa ukladajú len do formátu PDF a dajú sa hneď zdieľať.

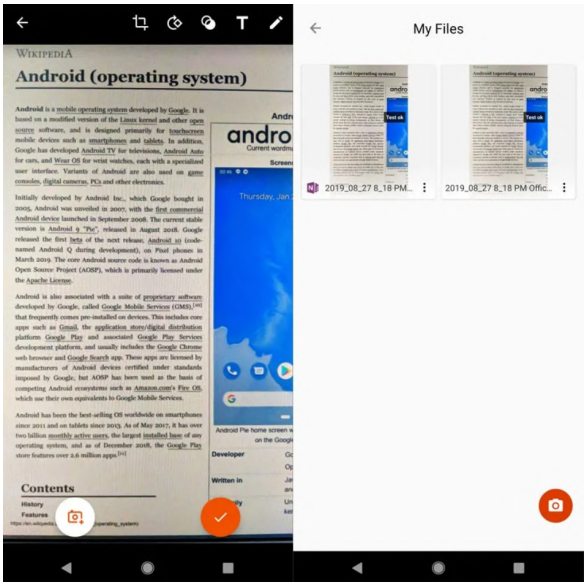


Obr. 2.2: **Adobe Scan,** zdroj[3]

Microsoft Office Lens

Táto aplikácia je skôr zameraná na spoluprácu s nástrojmi od Microsofu, ako je Microsot Word, Microsoft Powerpoint a ďalšie a je taktiež zdarma. Text z naskenovaného dokumentu dokáže exportovať priamo do Wordu. Dokáže rozpoznať firemné karty zamestnancov a uložiť ich ako nové kontakty do OneNote. Aplikácia má však horšiu kvalitu naskenovaných dokumentov. Je tu však lepšia možnosť exportu do rôznych formátov a dobré možnosti zdiaľania.

[3] https://www.androidinfotech.com/cam-scanner-android-apps/
[4] https://www.androidinfotech.com/cam-scanner-android-apps/



Obr. 2.3: **Microsoft Office Lens,** zdroj[4]

ScanPro

Jej základná verzia je zdarma, ale obsahuje veľký počet platených balíčkov. Ponúka však viac možností ako vyššie spomínané aplikácie, ako napríklad organizovanie súborov do priečinkov, automatické nahrávanie súborov do rôznych cloudových služieb alebo šifrovanie PDF dokumentov. Poskytuje OCR (optical character recognition - optické rozpoznávanie znakov) pre množstvo jazykov (v čase písania práce 60 jazykov pre iOS a 104 pre Android).

2.2 Desktopové aplikácie

Prekvapujúco, aplikácií zameraných na skenovanie a rýchlu úpravu dokumentov vôbec nie je veľa. Taktiež väčšina z nich ponúka podobné funkcie, ale nie také, ktoré by sa nám hodili v dosiahnutiu cieľa našej práce. Žiadna z nižšie uvedených aplikácií nepodporuje transformačné úpravy dokumentov, teda okrem Adobe Photoshop, ale to nie je program primárne určený na úpravy dokumentov. [3]

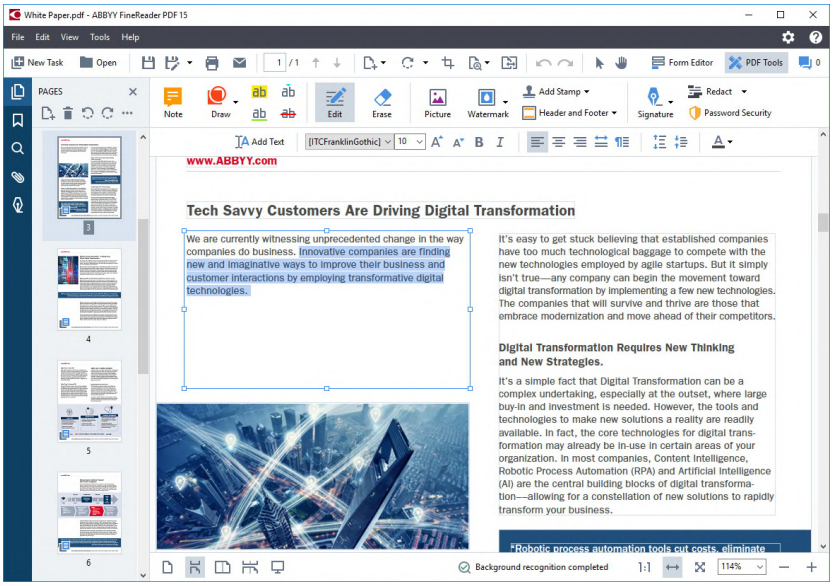
Abbyy FineReader

Platený software, ktorý je výborný na OCR. Dokáže otvoriť naskenované dokumenty a vytvoriť z nich PDF, v ktorom môžme vykonávať rôzne úpravy, či už textu, farieb, je možne pridávať komentáre, uchovávajú sa tu záznamy o zmenách, takže je možné sa vrátiť k predošlým úpravám. Jeho OCR podporuje až 192 jazykov. Nie je tam však možné vyrovnávať ohnuté strany dokumentov.

PaperScan

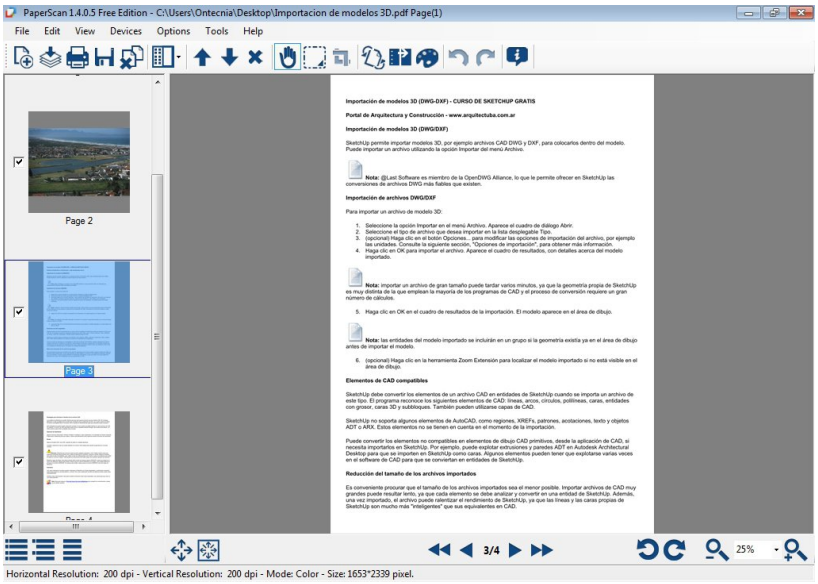
Tento software sa dá používať aj zadarmo, ale sú doň vložené reklamy. Tých sa dá zbaviť zakúpením plnej verzie softwaru, ktorá taktiež obsahuje viac funkcií. Už v základnej verzii

[5] https://pdf.abbyy.com/media/1715/02_finereader_15_edit_pdfs_en.jpg



Obr. 2.4: **Abbyy FineReader,** zdroj[5]

dokáže pracovať s naskenovanými dokumentmi, obsahuje rotáciu, orezanie okrajov, úpravu jasu, rôzne filtre, zmenu kvality snímky. OCR dokáže rozpoznať 60 jazykov. Nedokáže však vyrovnávať ohnuté strany.



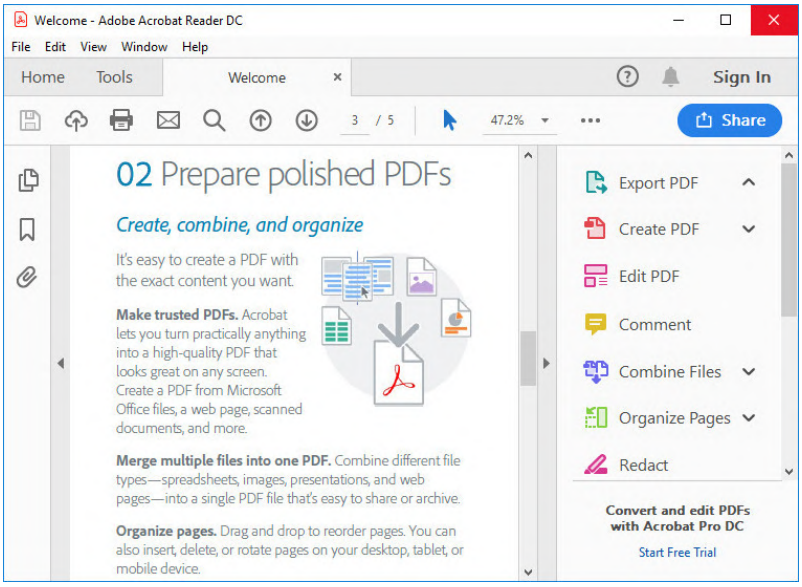
Obr. 2.5: **PaperScan,** zdroj[6]

Adobe Acrobat DC

Ďalši platený software. Jeden z najznámejších, čo sa týka práce s formátom PDF. Dokáže vytvoriť PDF z programov Microsoft Office, z odfotených snímok, z HTML stránok, emailov.

[6] https://www.malavida.com/en/soft/paperscan/

Podporuje zálohu na cloud, dokáže upravovať text aj tabuľky, konvertovať PDF dokumenty do Wordu, Powerpointu aj Excelu.



Obr. 2.6: **Adobe Acrobat DC,** zdroj[7]

Adobe Photoshop

Tento program je taktiež platený, ale je určený primárne na úpravu obrázkov, nie na prácu s dokumentmi. Je v ňom však možné vykonávať orezanie, rôzne transformácie, úpravy jasu a farieb a mnohé ďalšie operácie. Práca s ním však vyžaduje určitú prax, pretože sa nepoužíva najľahšie a niektoré úpravy môžu zabrať dlhší čas.

[7] https://www.neowin.net/news/adobe-acrobat-reader-dc-201900820071/

Kapitola 3

Spracovanie obrazu

3.1 Úvod do spracovania obrazu

Predtým, ako sa začneme venovať spracovaniu obrazu, je dôležité pochopiť širšie súvislosti, ktoré s týmto súvisia. Napríklad to, ako vníma obraz človek a ako je obraz realizovaný v počítači. Táto časť textu bola inšpirovaná zdrojom [6]

Človek dokáže vnímať svet okolo seba vďaka odrazu svetla od okolitých predmetov. Farba, ktorú vnímame, závisí od fyzikálnych vlastností objektu, od ktorého sa svetlo odráža. Viditeľné svetlo je súčasťou elektromagnetického žiarenia a má vlnovú dĺžku od 380 do 720 nm. Ľudské oko obsahuje dva druhy svetlocitlivých buniek - tyčinky a čapíky. Tyčinky vnímajú intenzitu svetla, sú veľmi citlivé a umožňujú nočné videnie. Čapíky rozpoznávajú svetlo rôznych vlnových dĺžok a umožňujú farebné videnie. Obidva druhy buniek sa nachádzajú na sietnici oka.

Počítač však nedokáže vnímať farbu. Obraz môžme v počítači definovať dvoma spôsobmi: rastrovo alebo vektorovo.

Vektorová grafika

Vektorová grafika je definovaná matematicky, skladá sa z geometrických tvarov (body, pramky, krivky). Ich veľkosť a vzdialenost sú vypočítané na základe rovníc. Vďaka tomu obrázky nestrácajú kvalitu, ale nie sú vhodné na zobrazovanie fotografií.

Rastrová grafika

Rastrová grafika nás bude v tejto práci zaujímať viac. Na zobrazenie nepoužíva rovnice, ale každý obrázok je rozdelený do bodov - pixelov (z anglického picture element, teda obrazový bod). Obrázky v rastrovej grafike majú určitú výšku a šírku, to sa určuje počtom pixelov. Každý pixel sa dá jednoznačne identifikovať na základe jeho súradníc. Ak v sebe nesie len jednu informáciu, vznikne nám monochromatický obraz. Ak chceme, aby bol obraz farebný, potrebujeme mať v každom pixeli uložených viac informácií. Najskôr si však musíme vybrať farebný model a hĺbku farby.

Farebný model

Obsahuje základné farby a ich miešaním dostávame ostatné farby farebného spektra. Medzi najznámejšie farebné modely patrí RGB(A). Zákad tohto modelu tvoria 3 farby, konkrétne červená, zelená a modrá. Je to aditívny farebný model, takže farby vznikajú pridávaním

troch základných farieb. To znamená, že ak nepridáme žiadnu farbu, budeme mať čiernu a ak zmiešame všetky farby, dostaneme bielu farbu. Písmeno A v RGBA znamená alfa, týmto sa dá upraviť priehľadnosť. Ďalším farebným modelom je CMYK. Tento model je substraktívny, čiže farby vznikajú odoberaním zložiek - tyrkysovej (Cyan), fialovej (Magenta), žltej (Yellow). Model je doplnený čiernou (blacK). RGB model sa používa na zobrazenie na monitoroch, zatiaľ čo CMYK sa využíva skôr na tlač. Ďalšími modelmi sa v tejto práci nebudeme zaoberať.

Farebná hĺbka

Používa sa na označenie počtu bitov, pomocou ktorých je popísaná farba. Väčšia farebná hĺbka zvyšuje škálu farieb, ale taktiež zvyšuje pamäťovú náročnosť obrázku.

Spracovanie obrazu

Spracovanie obrazu je spôsob spracovania signálu, na vstupe ktorého sú obrazové dáta. Pri spracovaní obrazu sa snažíme transformovať pôvodný obraz na nami požadovaný obraz. Konkrétne spracovanie obrazu sa skladá z niekoľkých častí, ktoré sú popísané v tejto kapitole.

3.2 Digitalizácia obrazu

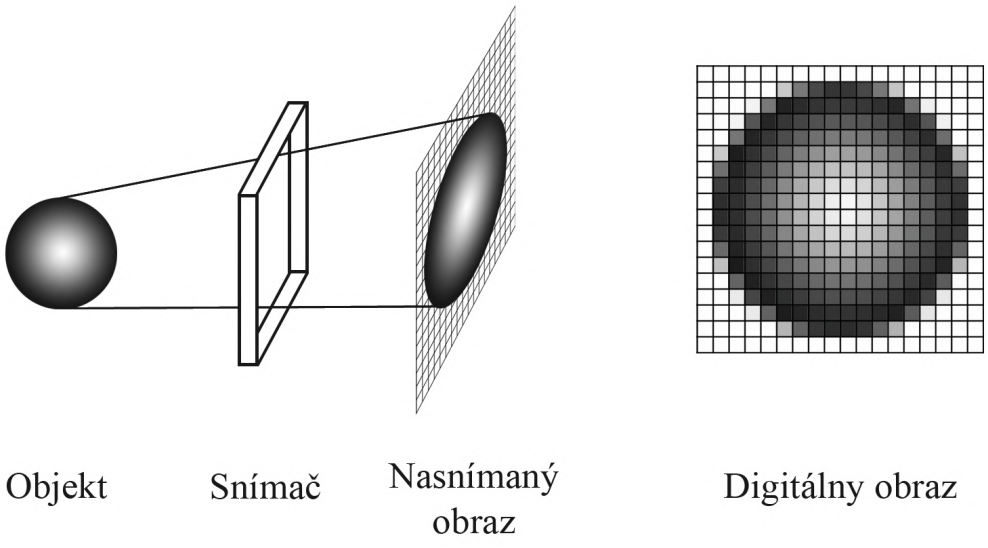
Táto časť textu bola inšpirovaná zdrojom [5]. Prvým krokom pre spracovanie a rozpoznávanie obrazu je získanie obrazu reálneho sveta, jeho prevod do digitálnej formy vhodnej pre uloženie a jeho ďalšie spracovanie. Všetko začína snímaním, teda prevodom optickej veličiny na spojitý elektrický signál. Ďalším krokom je prevod analógového signálu na digitálny - digitalizácia.

Digitálny obraz je ekvivalentom spojitej obrazovej funkcie f(i,j), kde i a j sú súradnice v priestore. Je získaný pomocou vzorkovania obrazu (diskretizacie súradníc obrazu) do matice o veľkosti M x N a kvantovania (diskretizácie jasových úrovní), to znamená rozdelenie spojitej jasovej úrovne obrázku do K úrovní. Čím je vzorkovanie a kvantovanie jemnejšie (väčšie M a N a väčší počet jasových úrovní), tým je aproximácia pôvodného spojiteho obrazového signálu lepšia.

Otázku vzdialenosti vzoriek (vzdialenosť medzi najbližšími vzorkovanými bodmi v obraze) rieši **Shannonova veta**:

- Pre jednorozmerné signály musí byť vzorkovacia frekvencia minimálne dvakrát väčšia ako najvyššia frekvencia vo vzorkovanom signáli, aby bolo možné signál spetne rekonštruovať
- Pre dvojrozmerné (obrazové) signály musí byť interval vzorkovania minimálne dvakrát menší, ako veľkosť najmenšieho detailu v obraze, pre optimálne spracovanie obrazového signálu sa najčastejšie používa veľkosť vzorkovaného elementu 5-krát medza daná vzorkovacou vetou, takže vzorkovacia frekvencia je 1/5 veľkosti najmenšieho detailu

Počet kvantovacích úrovní musí byť dostatočne veľký, aby boli presne vyjadrené detaily obrazu, nevznikali falošné obrysy a aby sa citlivosť snímacieho zariadenia blížila citlivosti ľudského oka. Ľudský zrak je schopný rozlíšiť cca 50 úrovní jasu v monochromatickom



Obr. 3.1: **Digitalizácia obrazu**

obraze. Pri menšom počte začne človek vnímať falošné obrysy. Väčšinou sa používa kvantovanie do 8 bitov, niekedy stačí do 6, alebo 4. Počet jasových úrovní K dostaneme dosadením do vzorca:

$$K = 2^b \tag{3.1}$$

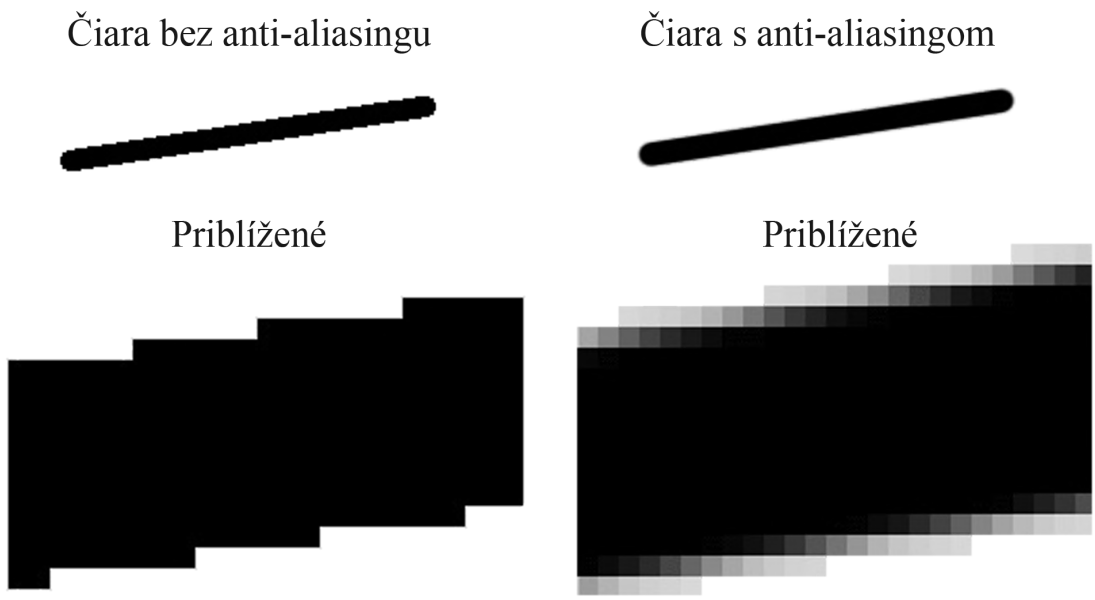
kde b je počet bitov.

3.3 Predspracovanie obrazu

Predspracovanie obrazu je dôležitou časťou celkového spracovania obrazu. Dáta (snímky), s ktorými potrebujeme pracovať prichádzajú z rôznych zdrojov, majú iné rozlíšenie, rôzny šum, rôzne farby. Na spracovanie obrazu sa využívajú rôzne algoritmy a aby sme zaistili, že budú pracovať čo najpresnejšie, musíme vstupné dáta upraviť do požadovanej formy. Nemôžme predsa pre každú snímku upravovať používané algoritmy. V tejto časti sa zameriame na techniky, ktoré sa používajú na predspracovanie obrazu. [11]

Zmena veľkosti

Vo vektorovej grafike to nie je problém, tam sa nestráca kvalita. Nás však zaujíma rastrová grafika. Keďže je tvorená maticou, teda pevným počtom pixelov, pri zmene veľkosti sa určité informácie strácajú. Proces, pri ktorom sa mení veľkosť snímky sa nazýva prevzorkovanie (anglicky resampling), pri ktorom sa pixely mapujú do novej matice, ktorá môže byť väčšia alebo menšia ako pôvodná matica. Zmenšovanie snímky môže viesť k strate kvality, keďže sa pixely mapujú do menšej matice. Farby musia byť premiešané a často upravené procesom, ktorý sa nazýva anti-aliasing (vyhladzovanie). Pri použití dobrého algoritmu je väčšina detailov zachovaná, ale čím viac sa zmenšuje, tým viac detailov sa stráca. [2]



Obr. 3.2: **Príklad aliasingu a anti-aliasingu**

Pri zväčšovaní sa taktiež stráca kvalita, snímka vyzerá byť rozmazaná. Taktiež sa môže použiť anti-aliasingový filter, ale snímka bude pravdepodobne aj tak rozmazaná. Nasleduje porovnanie často používaných metód na zväčšovanie snímok [13]:

Nearest-neighbor interpolation

Jedna z jednoduchších metód. Nahradzuje jeden pixel viacerými pixelmi rovnakej farby. Výsledný obraz je zväčšený a zachováva všetky detaily, ale vyzerá veľmi hranato.

Bilinear interpolation

Využíva lineárnu interpoláciu, ale v dvoch rozmeroch. Fuguje tak, že pre každý riadok a následne každý stĺpec matice interpoluje hodnotu farieb pixelov a kontinuálne ich prevádza na výstup. Táto metóda znižuje kontrast - odstraňuje ostré hrany.

Image tracing

Metóda najskôr zvektorizuje snímku, ktorá sa má zväčšiť. Potom sa zvektorizovaná snímka vykreslí ako rastrový obrázok v požadovanom rozlíšení. Metóda je vhodná na geometrické obrazce ako je text, ale nie je vhodná na fotografie.

Deep convolutional neural networks

Pomocou strojového učenia sa generujú detaily obrázkov a to na základe odhadu, ktorý je získaný z tréningového súboru. Výsledok môže niekedy pôsobiť zvláštne a nemusí odpovedať pôvodnému obrazu.

[1] https://en.wikipedia.org/w/index.php?title=Comparison_gallery_of_image_scaling_algorithms&oldid=989411683

Obr. 3.3: **Porovnanie metód zväčšovania snímok,** zdroj[1]

Zmenšovanie snímok

Na zmenšovanie snímok sa taktiež používajú techniky ako Billinear interpolation, Image tracing alebo Deep convolutional neural networks, pri ktorých sa však prevádzajú pixely z väčšej rastrovej mriežky do menšej.

Redukcia farebného priestoru

Vo väčšine prípadov pracujeme so snímkami, ktoré využívajú niektorý z farebných modelov spomínaných vyššie, to znamená, že majú viacero farebných kanálov. Ak je snímka uložená v RGB a využíva farebnú hĺbku 8 bitov, tak dokážeme vypočítať, že jedna farba je defonovaná pomocou 24 bitov. Z toho nám výjde, že v snímke sa môže nachádzať vyše 16 miliónov farieb. To sa nám vzhľadom na veľkosť dát a použité algoritmy veľmi nehodí. Preto sa snímka upravuje pomocou nasledujúcich techník: [11]

GrayScale

Táto technika redukuje farebnú snímku na snímku, ktorá sa skladá zo stupňov šedej. Tie sú väčšinou vzorkované na 8 bitov, takže dostaneme 256 stupňov šedi. Na prevod sa používa vzťah:

$$I = 0.299R + 0.587G + 0.114B \tag{3.2}$$

Následné metódy už využívajú snímky v stupňoch šedi.

Dithering/Halftoning

Po tom, ako dostaneme stupne šedi, môžme využiť nasledujúce metódy, aby sme získali snímku obsahujúcu len dve farby - čiernu a bielu. Obe metódy nahradzujú pixely za čiernobiele tak, aby nová snímka dosiahla vizuálne odpovedajúcu podobu pôvodnej snímky. Rozdiel je v tom, že dithering nezväčšuje rozlíšenie snímky (význam pri zobrazení na monitoroch s obmedzeným rozlíšením), zatiaľ čo halftoning áno (význam pri tlači).

Prahovanie

Táto metóda taktiež slúži na prevod snímky v stupňoch šedi na čierno-bielu. Je založená na porovnávaní všetkých pixelov s určitou prahovou hodnotou. pixely, ktorých hodnota je pod prahovou hodnotou budú čierne, ostatné budú biele. Táto metóda je veľmi rýchla,

ale má dosť deštruktívne účinky na snímky. Pri tejto metóde by som spomenul aj metódu náhodného rozptýlenia. Tiež funguje na princípe prahovania, ale prahová hodnota sa pre každý pixel náhodne generuje. Metóda dosahuje lepšie výsledky pre snímky s veľkými konštantnými plochami.

Maticové rozptýlenie

Metóda je založená na porovnávaní pixelov obrazu s odpovedajúcimi hodnotami vhodnej distribučnej matice. Pixely s hodnotou menšou ako hodnota matice budú čierne, ostatné budú biele. Túto metódu je možné aplikovať ako dithering (veľkosť snímku sa nemení) alebo halftoning (každý pixel je rozdelený do viacerých pixelov).

Distribúcia chyby

Metóda je založená na metóde prahovania s tým, že vzniknutá chyba je distribuovaná okolitým, ešte nespracovaným pixelom. Ak je pixel prevedený na čierny, chyba sa rovná pôvodnej hodnote pixelu, ak je pixel prevedený na biely, chyba sa rovná rozdielu maximálnej možnej hodnoty pixelu a pôvodnej hodnoty.

Obr. 3.4: **Porovnanie metód redukcie farebného priestoru,** zdroj [11]

3.4 Detekcia hrán

Detekcia hrán je základnou časťou spracovania obrazu, počítačového videnia, detekcie a extrakcie prvkov obrazu. Zahŕňa veľké množstvo matematických metód, ktoré sa snažia detekovať body, v ktorých sa prudko mení intenzita jasu v digitálnom obraze. Tieto body sa zhlukujú do zakrivených línií, ktoré sa nazývajú hrany. Hrany môžu byť výsledkom zmeny svetla, farby, tvaru, textúry. Detekcia hrán teda môže byť definovaná ako hľadanie čiar, ktoré vyznačujú hranicu medzi rôznymi prvkami v obraze. Aplikovanie algoritmov na detekciu hrán snímky môže do značnej miery zredukovať množstvo dát, ktoré je potrebné spracovať, teda dokáže vyfiltrovať menej podstatné informácie zo snímky, zatiaľ čo ponecháva pre nás

dôležité informácie. Nie je to však jednoduchá činnosť, pretože pri nej dochádza k fragmentácií, čo znamená, že jednotlivé hrany nie sú prepojené a taktiež sú detekované falošné hrany. [9]

Prístupy k detekcii hrán

Metódy detekcie hrán sa delia na dve kategórie: založené na gradiente (angl. gradient based) a založené na Laplacovom operátore (angl. Laplacian based). Gradient - smerová zmena intenzity farby obrázka.

Metóda založená na gradiente detekuje hrany na základe výpočtu prvej derivácie zo snímky. Hrany sa určujú na základe lokálneho maxima zo sklonu tejto derivácie. V mieste najväčšej zmeny hodnoty obrazovej funkcie dosahuje prvá derivácia najvyššie hodnoty. Metóda založená na Laplaceovom operátore detekuje hrany na základe výpočtu druhej derivácie, konkrétne kde sa rovná nule (zero crossing). Keďže je Laplaceov operátor citlivý na šum, typickou prípravou na detekciu hrán je použitie Gaussovho vyhladenia (angl. Gaussian blur).

Gaussovo vyhladenie

Gaussové vyhladenie je výsledok rozmazania obrazu použitím Gaussovej funkcie. Používa sa na redukciu šumu v snímke. Vykonáva sa pomocou konvolúcie medzi vstupnou snímkou a Gaussovským konvolučným jadrom. Ide o priemerovanie hodnôt pixelov z okolia, pričom väčšia váha sa kladie na pixely blízko stredu. Vyhladzovanie rozmazáva ostré hrany v snímke. Pre jadro musia platiť tieto vlastnosti: [14] [12]

- Veľkosť jadra musí byť nepárne číslo
- Súčet všetkých prvkov musí byť 1

Pre dvojrozmernú maticu sa hodnoty vypočítajú ako:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.3}$$

kde σ je smerodajná odchýlka, ktorá udáva strmosť.
Príklad jadra o veľkosti 5x5:

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix} \tag{3.4}$$

Metódy

Nasledujúce informácie boli čerpané z [9] [8]. Gradient vektoru ∇f sa počíta následovne:

$$\nabla f = G[F(x,y)] = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{pmatrix} \frac{df}{dx} \\ \frac{df}{dy} \end{pmatrix} \tag{3.5}$$

Obr. 3.5: **Gaussovo vyhladenie**

Veľkosť gradientu:

$$|G| = \sqrt{G_x^2 + G_y^2} \tag{3.6}$$

Smer gradientu:

$$\theta = \arctan(G_y, G_x) \tag{3.7}$$

Robertov operátor

Robertsnov detektor hrán bol jeden z prvých detektorov hrán a bol predstavený Lawrencom Robertsnom v roku 1965. Vďaka malému jadru a použitiu celých čísel vykonáva rýchly výpočet priestorového gradientu obrázku. Vstupom detektoru je obrázok v stupňoch šedej, hodnoty pixelov výstupného obrázku predstavujú odhadovanú veľkosť priestorového gradientu v daných bodoch. Táto metóda je však veľmi citlivá na šum. Vo vzorci 3.8 je vidieť príklad jadier (A je vstupný obrázok).

$$G_x = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} * A \qquad G_y = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} * A \tag{3.8}$$

Sobelov operátor

Vyvinutý Irwinom Sobelom a Garym Feldmanom v roku 1968. Je podobný Robertovmu operátoru, ale používa jadrá o veľkosti 3x3, kde druhé sa rovná prvému otočenému o 90 stupňov. Používa sa na nájdenie absolútnej veľkosti gradientu v každom bode na obrázku v stupňoch šedej. Príklad jadier je vo vzorci 3.9

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * A \qquad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A \tag{3.9}$$

Prewittin operátor

Podobne ako Sobelov operátor, aj Prewittin operátor používa jadro o veľkosti 3x3, používa celé čísla pre rýchlejší výpočet, ale výsledky sú hrubé, neopracované, hlavne pre obrázky s väčším počtom zmien (štruktúr, textúr).

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} * A \qquad G_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} * A \tag{3.10}$$

Cannyho algoritmus

Bol vytvorený Johnom Cannym ako súčasť jeho diplomovej práce na MIT v roku 1983 a stále dosahuje lepšie výsledky, ako novšie algoritmy. Je jednou zo štandardných metód v priemysle. Pri tejto metóde je však veľmi dôležité zbaviť sa šumu, napríklad Gaussovým vyhladením, pretože je naň veľmi citlivá, na rozdiel od Robertsnovej a Sobelovej metódy. Požiadavky pri detekcii hrán:

- **minimálny počet chýb**: musia byť detekované všetky hrany, nesmú byť detekované miesta, kde hrany nie sú,
- **presnosť**: poloha hrany musí byť určená čo najpresnejšie,
- **jednoznačnosť**: odozva na hranu musí byť len jedna, nesmie dochádzať ku zdvojeniu v dôsledku šumu.

Cannyho detektor hrán používa štyri filtre na detekciu horizontálnych, vertikálnych a diagonálnych hrán. Využíva Robertsov, Sobelov alebo Prewittin operátor, ktoré vypočítajú veľkosť prvej derivácie v horizontálnom a vertikálnom smere. Proces Cannyho algoritmu detekcie hrán môže byť popísaný v niekoľkých krokoch:

1. Aplikovanie Gaussovho rozostrenia na odstránenie šumu
2. Určenie gradientu na základe prvej derivácie
3. Potlačenie falošných bodov vzniknutých pri detekcii
4. Použitie prahovania na odhalenie potenciálnych hrán

Vzorcom 3.3 sa vypočíta konvolučná maska, ktorá je aplikovaná na celý obraz. Na výpočet veľkosti a smeru gradientu je výhodné použiť Sobelov operator, ktorý nie je príliš citlivý na šum. Falošné body sa potlačia tak, že sa vyberajú len lokálne maximá hodnoty gradientu, respektíve sa odoberú body, v ktorých nie je maximum. Týmto zaistíme čisté línie. Je však ešte potrebné vykonať prahovanie. To sa vykonáva pomocou dvoch prahových hodnôt: väčšej a menšej. Ak je gradient bodu väčší ako väčšia prahová hodnota, je bod označený ako silný. Ak má gradient hodnotu menšiu ako väčšia prahová hodnota, ale väčšiu ako menšia prahová hodnota, je bod označený ako slabý a ak je hodnota gradientu menšia ako menšia prahová hodnota, je bod odstránený. Následne sa preveria slabé body, ktoré môžu označovať správne hrany, ale taktiež mohli vzniknúť v dôsledku šumu. To sa vykonáva tak, že sa prehľadá okolie slabého bodu (8-okolie) a akonáhle sa v tomto okolí nachádza silný bod, tak je slabý bod ponechaný, v opačnom prípade je odstránený. [10]

Detekcia hrán je dôležitou časťou spracovania obrazu, preto je dôležité poznať rôzne techniky súvisiace s detekciou hrán. V texte boli spomenuté metódy založené na gradiente. Robertsnov operátor je veľmi citlivý na šum a neprodukuje dostatočné výsledky, je však veľmi rýchly na počítanie. Sobelov operátor je menej citlivý na šum, ale je pomalší ako Robertsnov operátor. Výsledky Prewittinho operátora sú zase nedostatočné a je potrebné vykonávať dodatočné spracovanie. Cannyho algoritmus detekcie hrán je menej citlivý na šum ako spomínané metódy, ale je náročnejší na výpočet. Dosahuje však lepšie výsledky.

3.5 Transformácia obrazu

Zdrojom tejto podkapitoly je [4]. Geometrická transformácia obrazu je potrebná pre korekciu akýchkoľvek geometrických skreslení vzniknutých počas snímania obrazu, alebo na vytvorenie geometrických efektov. V oboch prípadoch musí byť transformácia schopná reprodukovať obraz s rovnakými informáciami. Geometrická transformácia modifikuje pixely v obraze. To je dosiahnuté použitím vhodných matematických metód, ktoré transformujú koordináty (x,y) vstupného obrázku na koordináty (x',y') výstupného obrázku. V kontexte digitálnych obrazov sa geometrická transformácia skladá z dvoch druhov operácií.

- **Geometrické operácie**: vykonáva priestorové transformácie pixel po pixeli medzi súradnicami (x,y) vstupného obrázku a súradnicami (x',y') výstupného obrázku.
- **Interpolačné operácie**: každému pixelu výstupného obrázku priradí príslušnú úroveň šedej na základe interpolácie hodnoty pixelu vstupného obrázku.

Geometrické operácie

Matematický vzťah popisujúci vzťah súradníc pixelov vstupného obrazu s výstupným obrazom je možné definovať všeobecne pomocou nasledujúcej rovnice:

$$g(x,y) = f(x',y') = f[T_x(x,y), T_y(x,y)] \tag{3.11}$$

kde (x',y') sú nové súradnice pixelu pôvodného obrázku so súradnicami (x,y), zatiaľ čo funkcie T_x (x, y) a T_y (x, y) jedinečne určujú priestorové transformácie aplikované na pixely vstupného obrazu f(x, y).

Funkcie priestorovej transformácie sú dané nasledujúcimi všeobecnými rovnicami:

$$x' = T_x(x,y) \qquad x = T_x^-1(x',y') \tag{3.12}$$

$$y' = T_y(x,y) \qquad y = T_y^-1(x',y') \tag{3.13}$$

V ľavej časti rovníc 3.12 a 3.13 sa nachádzajú rovnice, ktoré vyrátajú novú pozíciu pixelu x a y, zatiaľ čo na pravej strane sa nachádzajú inverzné rovnice, ktoré nám z novo vzniknutého pixelu na pozícií x' a y' vrátia pôvodnú pozíciu pixelu. Geometrická transformácia môže byť lineárna a nelineárna:

- **Lineárna**: zahŕňa jednoduché transformácie, ako sú zväčšenie, zmenšenie a rotácia
- **Nelineárna**: okrem lineárnych transformácií využívajú aj transformácie na opravu zakrivenia

Zväčšenie a zmenšenie

Transformačné rovnice na zväčšenie a zmenšenie obrázku sú dané následovne:

$$x' = x \cdot S_x \tag{3.14}$$
$$y' = y \cdot S_y \tag{3.15}$$

Ak je $S_x > 1$ a $S_y > 1$, výsledkom bude zväčšenie. Ak je $S_x <1$ a $S_y < 1$, výsledkom je zmenšenie a ak je $S_x = 1$ a $S_y = 1$ ´, nedochádza k zmene veľkosti. Inverzná transformácia je daná vzťahom:

$$x = x'/S_x \tag{3.16}$$
$$y = y'/S_y \tag{3.17}$$

Rotácia

Obrázok môže byť otočený o uhol θ okolo počiatku súradnicového systému. To znamená, že pre každý pixel pôvodného obrázku sa vypočíta jeho nová pozícia vzhľadom na osi x a y. Na obrázku 3.6 je možné vidieť bod P na pozícií (x,y), ktorý je reprezentovaný vektorom $\vec{v}$ s veľkosťou ρ a od osi x je odchýlený o uhol α. Rotáciou bodu P okolo počiatku súradnicového systému o uhol θ dostaneme bod P' s koordinátmi (x',y'). Vzťah medzi P a P' sa dá zapísať ako:

$$x = \rho\cos\alpha \qquad x' = \cos\alpha + \theta \tag{3.18}$$
$$y = \rho\sin\alpha \qquad y' = \sin\alpha + \theta \tag{3.19}$$

Výslednú pozíciu bodu P' je možné vypočítať pomocou rovníc:

$$x' = x \cdot \cos\theta - y \cdot \sin\theta \tag{3.20}$$
$$y' = x \cdot \sin\theta - y \cdot \cos\theta \tag{3.21}$$

Zošikmenie a skosenie

Táto operácia predstavuje posun pixelov o uhol θ okolo jednej osi, hodnoty druhej osi zostanú nezmenené. Zošikmenie podľa osi x a osi y je definované ako:

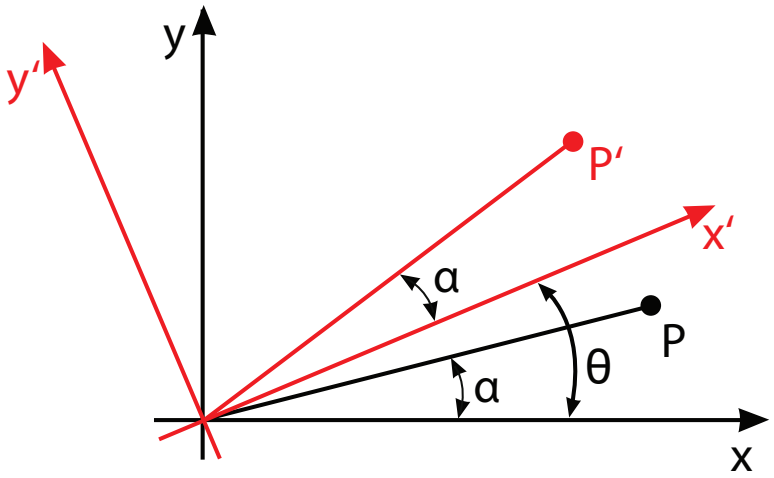
$$x' = x + y\tan\theta = x + sh_x \cdot y \qquad y' = y \tag{3.22}$$
$$y' = y + x\tan\theta = y + sh_y \cdot x \qquad x' = x \tag{3.23}$$

kde veľkosť zošikmenia môže byť daná konštantami sh_x pre smer osi x a sh_y pre smer osi y.

Homogénne geometrické transformácie

Geometrické transformácie sa nazývajú homogénne, keď je vstup aj výstup vyjadrený v homogénnych súradniciach. Operácie spomínané vyššie sa dajú zapísať v matici

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \boldsymbol{T} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.24}$$



Obr. 3.6: **Rotácia**

kde $\boldsymbol{T}$ je matica o veľkosti 3x3 s konkrétnou operáciou. Napríklad otočenie v homogénnych súradniciach môžme zapísať ako:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & -\cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.25}$$

Afinné geometrické transformácie

Afinná geometrická transformácia je taká, ktorá zachováva rovnobežnosť a deliaci pomer. Vo všeobecnosti, lineárne transformácie popísané v 3.11 sú afinné. Napríklad zväčšovanie sa vykonáva vložením hodnôt S_s a S_y do afínnej matice, čoho výsledkom je:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.26}$$

Rovnako ako pre 3.14, aj tu platí, že ak je S_x a $S_y > 1$, ide o zväčšenie, ak je S_x a $S_y < 1$, ide o zmenšenie a ak sú S_x a $S_y = 1$, nenastane žiadna zmena. Parametre S_x a S_y však môžu byť rôznych hodnôt. V takom prípade dochádza ku zmene tvaru objektu, napríklad zo štvorca sa stane obdĺžnik.

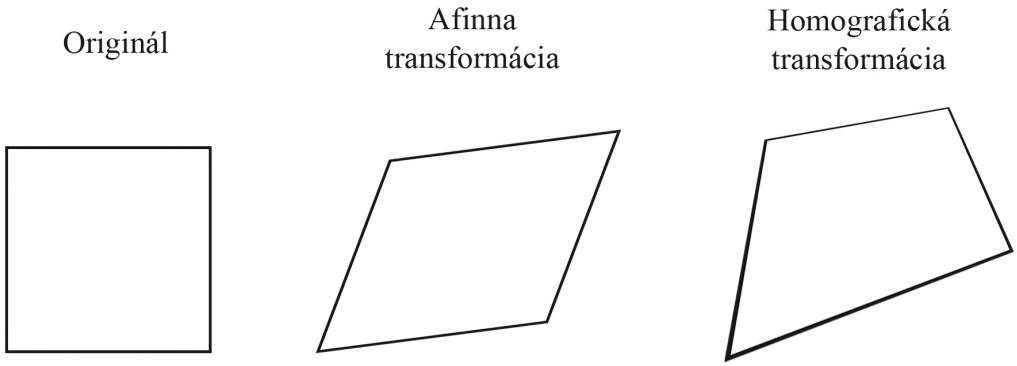
Afínna matica môže byť realizovaná kombináciou viacerých transformáci, napríklad:

$$\boldsymbol{A} = \boldsymbol{R} \times \boldsymbol{S} \tag{3.27}$$

kde $\boldsymbol{R}$ je rotácia, $\boldsymbol{S}$ je zmena veľkosti a $\boldsymbol{A}$ je výsledná afínna matica.

Homografická transformácia

Homografická transformácia súvisí s transformáciou medzi dvoma rovinami. Spája bod P(x,y,1) jedného plátna s bodom P'(x',y',1) z druhého plátna.



Obr. 3.7: **Afinná a homografická transformácia**

3.6 Úprava jasu

Zdrojom tejto podkapitoly je [1]. Jas je atribút vizuálneho vnímania, pri ktorom sa zdá, že zdroj vyžaruje alebo odráža svetlo. Je to subjektívny vnem, momentálne neexistuje jeho presný matematický popis. Najlepšie sa pracuje s úpravou jasu v dvoch modeloch: HSV (Hue, Saturation, Value) - (odtieň, sýtosť, jas), alebo inak HSB (Hue, Saturation, Brightness) alebo BCH (Brightness, Chroma, Hue) - (jas, sýtosť, odtieň). Model HSV prevláda v algoritmoch úpravy sýtosti a odtieňa. BCH je vhodný pre algoritmy, ktoré vykonávajú len požadovanú úpravu bez neželanej úpravy ďalších obrazových parametrov.

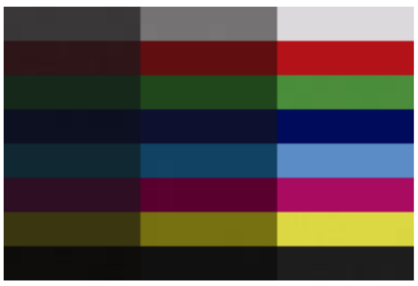
Natural choice algoritmus

Tento algoritmus vyzerá byť ako najprirodzenejší v úprave jasu. Je vyjadrený vzorcom

$$B' = 2^E V \cdot B \tag{3.28}$$

Algoritmus je navrhnutý pre model BCH, ale môže byť použitý aj pre iné farebné koordinačné systémy (Color Coordinate Systems - CSSs). Poskytuje lepšie výsledky ako algoritmus popísaný nižšie. V nasledujúcom obrázku sú zobrazené výsledky pre RGB.

Color	Original color			EV = +2			EV = +4		
Grey	56	56	56	111	111	111	208	208	208
Red	56	5	3	111	18	12	208	43	32
Green	2	37	15	8	77	38	25	148	78
Blue	5	4	29	18	15	63	43	38	123
Cyan	4	36	53	15	75	105	38	145	199
Magenta	54	2	28	107	8	61	202	25	119
Yellow	60	58	10	118	114	29	221	215	62
Dark	1	1	1	4	4	4	15	15	15



Obr. 3.8: **Výsledky Natural choice algoritmu,** zdroj [1]

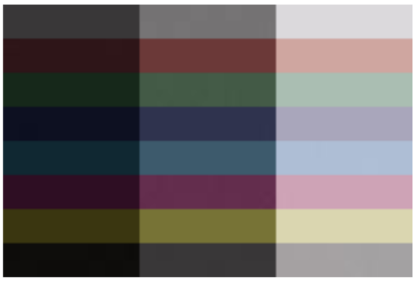
TV based algoritmus

Tento algoritmus napodobňuje nastavenie zmeny jasu v televízii. Využíva sa v mnohých nástrojoch spracovania obrazu, ako napríklad Corel alebo Photoshop. Na získanie novej farby sa využiva rovnica:

$$(r', g', b') = (r + M_0, g + M_0, b + M_0) \tag{3.29}$$

kde M_0 je parameter udávajúci zväčšenie/zmenšenie jasu. Výsledky algoritmu sú zobrazené v obrázku 3.9

Color	Original color			M_0 = +55			M_0 = +152		
Grey	56	56	56	111	111	111	208	208	208
Red	56	5	3	111	60	58	208	157	155
Green	2	37	15	57	92	70	154	189	167
Blue	5	4	29	60	59	84	157	156	181
Cyan	4	36	53	59	91	108	156	188	205
Magenta	54	2	28	109	57	83	206	154	180
Yellow	60	58	10	115	113	65	212	210	162
Dark	1	1	1	56	56	56	153	153	153



Obr. 3.9: **Výsledky TV based algoritmu,** zdroj [1]

Z porovnania obrázku 3.8 a 3.9 je jasne vidno, ako vplýva rovnomerná zmena jasu na kontrast a sýtosť farieb. Na jednej strane je jas nemerateľnou vlastnosťou. Na druhej strane je však nutné nejako zapísať jeho hodnotu kvôli spracovaniu digitálneho obrazu. Pre jas neexistuje konvenčný zápis, preto je bežné, že nástroje na editáciu obrazu používajú viacero foriem zápisu. BCH model ako jednotka merania jasu je vhodný pre vývoj softvéru. Aj keď ani jeden z modelov presne nezodpovedá ľudskému vnímaniu jasu, BCH funguje celkom dobre pri úprave obrázkov.

Kapitola 4

Návrh riešenia

Prvým krokom pri tvorbe programu je návrh, ako ho budeme riešiť. Tento návrh vyplýva z požiadavok, ktoré boli popísané v zadaní práce. Je potrebné vytvoriť program, ktorý dokáže spracovávať vyfotené dokumenty, orezať ich, vyrovnať ohnuté strany a upraviť jas. Taktiež je potrebné, aby tento program vyzeral graficky čo najpríjemnejšie.

4.1 Python

Na tvorbu práce bol použitý jazyk Python 3 a to z nasledujúcich dôvodov: Je to objektovo orientovaný programovací jazyk, ktorý spája vlastnosti výkonného návrhu softwarových vlastností tradičných programovacích jazykov s flexibilitou a použiteľnosťou skriptovacích jazykov. Práca s ním je jednoduchá, produktívna, programy sa ľahko implementujú. Taktiež sa programy rýchlejšie vyvíjajú a to hlavne vďaka tomu, že je objektovo-orientovaný. To prispieva k jednoduchšej čitateľnosti kódu a hlavne je program do budúcna ľahšie udržiavateľnejší. Ďalšou výhodou je, že na spustenie programov napísaných v Pythone nie je potrebný prekladač, stačí mať na zariadení nainštalovaný interpret jazyka Python alebo prípadne vytvoriť .exe súbor. Ďalšou výhodou Pythonu je veľká komunita vývojárov, ktorá poskytuje silnú základňu zdieľaných vedomostí a podpory. Taktiež existuje nespočet voľne dostupných knižníc na rôzne použitia. V minulosti bolo menšou nevýhodou interpretovaných programovacích jazykov to, že ich výsledné programy boli pomalšie ako programy kompilovaných programovacích jazykov, ale táto nevýhoda sa pri dnešných možnostiach hardvéru a softvéru už stráca. Samozrejme, že pri veľmi náročných aplikáciách sú rýchlejšie programy napísané v programovacích jazykoch nižšej úrovne, ale pre jednoduché programy to nezohráva úlohu. Python je taktiež považovaný za jeden s najlepších jazykov pre tvorbu grafického užívateľského rozhrania (GUI). Má zabudované možnosti vývoja GUI, alebo je možné použiť rôzne voľne dostupné frameworky. Je ľahko prenositeľný a beží na akejkoľvek platforme: Windows, Unix, Linux. . . . Python komunikuje bez problémov takmer s hocičím. Ak zistíte, že sa dá nejaká časť programu spraviť efektívnejšie v inom jazyku, nie je s tým problém.

4.2 OpenCV

Na splnenie zadania práce je však potrebné Python rozšíriť o ďalšie knižnice, ktoré nám uľahčia implementáciu programu. Konkrétne sa tým myslí použitie knižnice OpenCV. Táto knižnica podporuje viacero programovacích jazykov, konkrétne C++, Java, Python a ďal-

šie. Je možné ju použiť na mnohých operačných systémoch, ako Windows, Linux, OS X, Android a iOS. OpenCV slúži na riešenie problémov, ktoré súvisia so spracovaním obrazu, ako napríklad detekcia objektov, tvarov, ľudských tvárí alebo zvierat z obrázkov alebo videa. OpenCV-Python je "zabalená" knižnica OpenCV, ktorá je napísaná v jazyku C++, a používa sa v jazyku Python.

OpenCV-Python využíva knižnicu Numpy, čo je vysoko optimalizovaná knižnica pre numerické výpočty. Všetky polia využívané v OpenCV-Python sú konvertované do polí v Numpy. Taktiež bude použitá knižnica Pillow.

4.3 PyQt

Pomocou vyššie popísaných knižníc je možné vytvoriť požadovaný program, ale veľmi dôležitou časťou zadania je aj vytvorenie rozhrania, pomocou ktorého bude užívateľ pracovať s programom. Na tento účel je vhodné použiť nástroj Qt, konkrétne verziu pre Python - PyQt. Qt je jedna z najpopulárnejších multiplatformových knižníc pre vytváranie programov s grafickým užívateľským rozhraním. Je napísaná v programovacom jazyku C++, ale dá sa použiť, podobne ako vyššie spomínané knižnice, aj v jazyku Python, C, Java.... Je taktiež multiplatformná. Má veľmi dobre spracovanú dokumentáciu a vývojové programy Qt Creator alebo Qt Designer. Aplikácie vytvorené pre grafické užívateľské prostredie používajú natívny vzhľad operačného systému, takže vytvorené programy sa vždy prispôsobia vzhľadu používateľského prostredia.

Na implementáciu bude použitý Python verzie 3.8.5, ktorý sa dá stiahnuť z oficiálnych stránok Python-u. Pre inštaláciu knižníc OpenCV, Numpy, Pillow a PyQt bude potrebný nástroj pip, ktorý je potrebné nainštalovať pri inštalácii Python-u. Po tom, čo je nainštalovaný Python aj pip, je potrebné ho aktualizovať príkazom `pip install -upgrade pip`. Následne je možné nainštalovať ostatné potrebné knižnice. Tie sa nainštalujú týmto spôsobom: `pip install opencv-python`. Počas inštalácie OpenCV sa nainštaluje aj knižnica Numpy, ak by sa tak nestalo, tak sa nainštaluje príkazom `pip install numpy`. Knižnicu Pillow je možné nainštalovať príkazom `pip install Pillow`. Poslednú knižnicu, PyQt, je potrebné nainštalovať príkazom `pip install PyQt5`. Na to, aby sme mohli naplno využiť túto knižnicu, doinštalujeme doplnkové nástroje príkazom `pip install PyQt5-tools`.

Konkrétny návrh aplikácie sa bude skladať primárne z dvoch častí: časťou zaoberajúcou sa spracovávaním dokumentov a časťou, ktorá bude riešiť užívateľské rozhranie. Pre prehľadnosť tieto časti nazveme Backend časť a Frontend časť. Obe časti budú implementované pomocou tried a ich metód popísaných v nasledujúcej kapitole.

Kapitola 5

Implementácia

Ako bolo spomenuté v predchádzajúcej kapitole, implementácia aplikácie bude mať dve časti. Najskôr sa zameriame na Backend časť zaoberajúcu sa prácou s obrázkami: ich otvorenie, príprava, orezanie, vyrovnanie ohnutých strán a úprava jasu. Zdrojom informácií v tejto časti boli stránky dokumentácie[1] pre OpenCV.

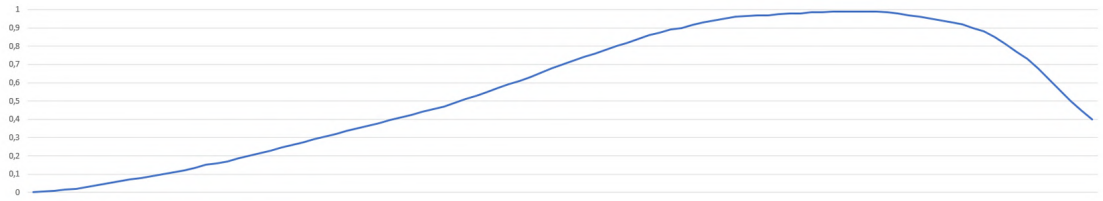
5.1 Backend časť

Táto časť programu je uložená v súbore s názvom `program.py`. Na začiatku súboru sú importované všetky potrebné knižnice, konkrétne OpenCV (cv2), numpy, math, regex (re), os, z knižnice PyQt5 Qt, QtCore, QtGui a QtWidgets a z knižnice PIL (Pillow) Image a ImageEnhance. Súbor obsahuje dve triedy: Program a AnotherWindow. Zatiaľ sa budeme zaoberať triedou Program. Na začiatku sú inicializované potrebné premenné:

- **width**: preddefinovaná šírka obrázku v pixeloch, implicitne 600,
- **height**: preddefinovaná výška obrázku v pixeloch, implicitne 800,
- **a4_width**: šírka výstupného obrázku v pixeloch, vypočíta sa z orezanej snímky,
- **a4_height**: výška výstupného obrázku v pixeloch, vypočíta sa z orezanej snímky,
- **two_pages**: premenná udáva, či sa na vstupnej fotografii nachádza jednostranný, alebo dvojstranný dokument,
- **left_page**: premenná udáva, či je na vstupnej fotografii vyfotená ľavá alebo pravá strana, táto premenná sa používa, ak premenná two_pages je nastavená na False,
- **show_files**: premenná udáva, či sa snímky po spracovaní zobrazia alebo nie,
- **manual**: premenná udáva, či bude užívateľ zadávať okraje dokumentov manuálne, alebo budú detekované automaticky,
- **top**: premenná určuje mieru zahnutia horného okraja strany, nadobúda hodnoty 0 až 5,
- **bottom**: premenná určuje mieru zahnutia dolného okraja strany, nadobúda hodnoty 0 až 5,

[1] https://docs.opencv.org/

- **out_folder**: premenná určuje umiestnenie priečinku, do ktorého sa budú ukladať výsledné obrázky,
- **colors**: premenná určuje typ operácie na úpravu farieb výsledného obrázku,
- **bend_arr**: pole o veľkosti 400 čísel s oborom hodnôt <0;1>, ktoré sa využíva na korekciu zahnutých okrajov strán. Ak si čísla nanesieme do grafu, výsledok je vidieť na obrázku 5.1.



Obr. 5.1: **Graf poľa bend_arr**

Ako je vidieť na obrázku 5.1, graf kopíruje ohnutie horného okraja strany, pri negácií hodnôt zase ohnutie dolného okraja strany.

Hlavná funkcia run()

Hlavnou funkciou tohto programu je funkcia `run(self, name)`, ktorá zabezpečuje celý chod Backend časti programu. Očakáva jeden argument `name`, ktorý obsahuje umiestnenie vstupného obrázku. Hneď na začiatku sa nachádza príkaz

```
img = cv2.imread(name)
```

, ktorý do premennej `img` uloží obrázok umiestnený tam, kam ukazuje obsah premennej `name`. Následne sa kontroluje, či sa obrázok načítal správne.

```
try:
    if img == None:
        return
except Exception:
    pass
```

Ak sa funkcii `cv2.imread()` nepodarí prečítať obrázok, funkcia vráti hodnotu `Null`, čiže premenná `img` bude prázdna, teda sa bude rovnať `None`. To sa môže stať napríklad, ak nie je súbor nájdený, čítanie je zamietnuté z dôvodu oprávnení, alebo nastal pokus o načítanie nepodporovaného formátu. Ak sa funkcia `cv2.imread()` vykoná úspešne, uloží obrázok do premennej `img` vo forme matice. Následne sa do premenných `original_height` a `original_width` uloží výška a šírka načítaného obrázku.

```
original_height, original_width, channels = img.shape
```

Ak je šírka obrázku väčšia ako jeho výška, je potrebné ho otočiť o 90° proti smeru hodinových ručičiek, je to potrebné pre správnu funkciu algoritmov použitých v ďalších častiach. Po otočení sa znova načíta výška a šírka obrázku.

```
if original_width > original_height:
    img = cv2.rotate(img, cv2.cv2.ROTATE_90_CLOCKWISE)
    original_height, original_width, channels = img.shape
```

Kvôli ďalšiemu použitiu si uložíme kópiu obrázku do premennej `copy_img` pomocou funkcie `copy()`. Ďalším krokom je zmena veľkosti obrázku. To sa vykonáva z dôvodu, že knižnica OpenCV vie lepšie pracovať s obrázkami s nižším rozlíšením. Taktiež to značne urýchli výpočet, pretože v ďalších častiach sa vykonávajú rôzne výpočty nad pixelmi a čím je ich menej, tým rýchlejší je výpočet. Zmenšený obrázok dostaneme pomocou funkcie `cv2.resize()`, ktorá prijíma pôvodný obrázok a výšku a šírku nového obrázku, ktorý vracia a uloží ho do premennej `resizedImg`. Následne sa na základe premennej `manual` určí, či budú dokumenty na snímke detekované automaticky, alebo ich užívateľ zadá manuálne. V prípade automatickej detekcie sa zavolajú funkcie na predspracovanie obrazu

Predspracovanie obrazu

Zavolá sa funkcia `preProcessing()`, ktorá prijíma zmenšený obrázok a vykonáva s ním nasledujúce operácie:

```
grayImg = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

Ako prvá sa použije funkcia `cv2.cvtColor()`, ktorá slúži na konvertovanie obrázku z jedného farebného spektra do iného. Táto funkcia umožňuje viac ako 150 možností konvertovania. V našom prípade bol použitý parameter `cv2.COLOR_BGR2GRAY`, ktorý prevedie obrázok `img` do odtieňov šedej a tento obrázok uloží do premennej `grayImg`. Tento obrázok sa následne použije vo funkcii `cv2.GaussianBlur()`.

```
blurImg = cv2.GaussianBlur(grayImg, (5, 5), 1)
```

Táto funkcia slúži na rozmazanie obrázku. Je to potrebné urobiť kvôli nasledujúcim funkciám na detekciu hrán. Funkcia prijíma vstupný obrázok, v tomto prípade `grayImg`, ďalej veľkosť jadra, ktoré musí mať určitú výšku a šírku. Tieto čísla sa nemusia zhodovať, ale je potrebné, aby boli obe nepárne. V tomto prípade je veľkosť jadra 5×5 pixelov. Ďalšími parametrami sú `sigmaX` a `sigmaY`, ktoré určujú odchýlku jadra v smere X a v smere Y. Ak nie je zadaná `sigmaY`, je rovná hodnote `sigmaX`. V tomto prípade je ich veľkosť 1. Táto funkcia zníži šum a odstráni rôzne menšie škvrny v obraze. Je dôležité ich odstránenie, pretože by v ďalších krokoch mohli spôsobiť detekciu falošných hrán. Ďalšou a veľmi dôležitou funkciou pri spracovaní obrazu je funkcia `cv2.Canny()`.

```
cannyImg = cv2.Canny(blurImg, 40, 150)
```

Táto funkcia prijíma ako prvý argument vstupný obrázok. Ďalším argumentom je minimálna prahová hodnota (`minVal`), pixely s nižšou hodnotou funkcie nie sú označené ako hrany. Tretí argument je maximálna prahová hodnota (`maxVal`), pixely s hodnotou funkcie väčšou ako je táto hodnota sú považované za hrany. Pixely, ktorých hodnota funkcie je menšia ako `maxVal` ale väčšia ako `minVal` sa ďalej skúmajú a zisťuje sa, či sú nejako prepojené s pixelmi označenými ako hrany. Ak áno, sú tiež označené ako hrany, v opačnom prípade nie sú označené. Počas vývoja aplikácie sa osvedčili hodnoty `minVal` 40 a `maxVal` 150. Ďalším argumentom je veľkosť jadra Sobelovho operátora, pomocou ktorého funkcia vypočítava gradient pixelov. Ak nie je zadaný, jeho veľkosť je 3. Aby sme zabezpečili detekciu dokumentov, samotný Cannyho detektor hrán nestačí. Môže sa stať, že je na snímke

dokument s pozadím, ktoré nie je v dostatočnom kontraste s dokumentom. V takom prípade sú odhalené len časti hrán a pre ďalšie použitie to nepostačuje. Preto sa v ďalšom kroku použije funkcia `cv2.HoughLinesP()`, ktorá využíva Houghovu transformáciu. Je to populárna metóda na detekciu tvarov, ktoré sa dajú zapísať matematicky, ako sú napríklad priamky. Dokáže ich detekovať aj keď sú poškodené alebo prerušované. Každá priamka sa dá popísať parametrami ρ - vzdialenosťou od stredu súradnicového systému a θ - uhlom voči ose x. Rovnica priamky v takom tvare je:

$$x\cos(\theta) + y\sin(\theta) = \rho \tag{5.1}$$

Pretože je každá priamka jednoznačne určená dvojicou (ρ, θ), každá priamka sa zobrazí zo súradnicového systému (x,y) do súradnicového systému (ρ, θ) ako jeden bod. Body x a y sú známe, ρ a θ sú neznáme. Dostaneme ich po dosadení do rovnice 5.1 tak, že budeme hodnotu θ iterovať na intervale $<0;\pi>$. V súradnicovom systéme (ρ, θ) sa nám tak vykreslí krivka. Pri zakreslení viacerých bodov ležiacich na jednej priamke sa nám vykreslia krivky, ktoré sa pretínajú v jednom bode. Na základe hodnoty ρ a θ tohto bodu vieme určiť umiestnenie a smer tejto priamky. Funkcia vyzerá nasledovne:

```
lines = cv2.HoughLinesP(cannyImg, 1, numpy.pi/180, 10,
    maxLineGap=40, minLineLength=200)
```

Jej prvým parametrom je čierno-biely (binárny) obrázok `cannyImg`, ktorý sme dostali z funkcie `cv2.Canny()`. Druhý parameter je ρ, veľkosť je 1, ďalší parameter je θ, jej hodnota je `numpy.pi/180`. Štvrtý parameter je prahová hodnota, ktorá určuje platnosť hrany. Parametrom `maxLineGap` sa určuje maximálny počet medzier medzi segmentami čiary, aby sa dala považovať za jednu čiaru. `minLineLength` určuje minimálnu dĺžku čiary. Čiary s menšou dĺžkou nie sú brané do úvahy. Výhodou funkcie `cv2.HoughLinesP()` je to, že nám vráti čiaru vrátane jej koncových bodov. Existuje taktiež funkcia `cv2.HoughLines()`, ale tá je náročnejšia na výpočet, pretože počíta všetky body. `cv2.HoughLinesP()` zasa využíva pravdepodobnostnú Houghovu transformáciu, takže nevypočítava všetky body, len náhodne vybrané. Výpočet je vďaka tomu oveľa rýchlejší, ale je potrebné znížiť prahovú hodnotu. Funkciu Houghovej transformácie je možné vidieť na obrázku 5.2

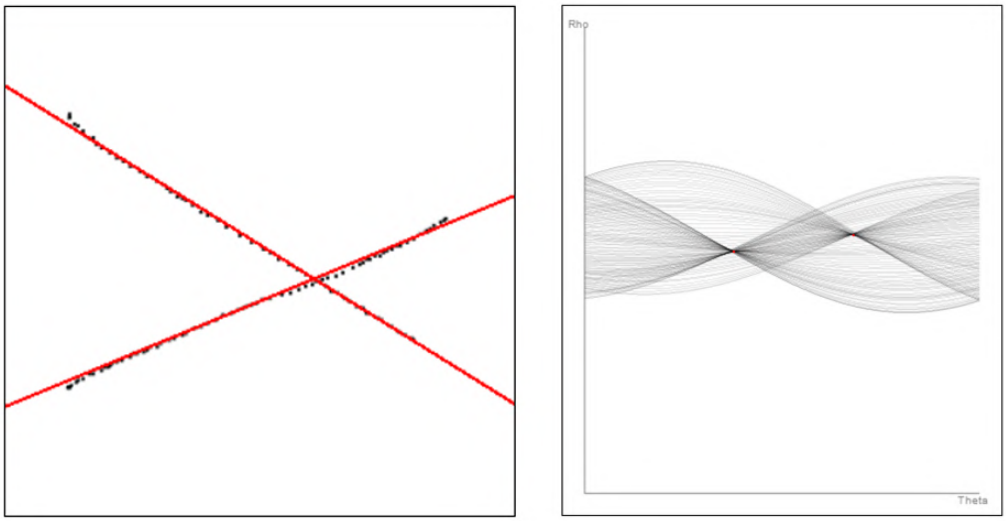
Funkcia `cv2.HoughLinesP()` nám vráti pole všetkých odhalených čiar, konkrétne ich začiatky a konce. Tieto čiary následne nanesieme do obrázku `cannyImg`.

```
for line in lines:
    x1,y1,x2,y2 = line[0]
    cv2.line(cannyImg, (x1,y1), (x2,y2), (255,255,255), 3)
```

Pomocou cyklu prechádzame pole `lines` a získame súradnice x a y pre začiatok a koniec každej čiary. Tie následne pomocou funkcie `cv2.line()` nanesieme do obrázku `cannyImg`. Aby sme zvýšili šancu na úspešnú detekciu dokumentu v snímke použijeme nasledujúce funkice:

```
kernel = numpy.ones((5, 5))
dilateImg = cv2.dilate(cannyImg, kernel, iterations=4)
threshImg = cv2.erode(dilateImg, kernel, iterations=2)
```

Premenná `kernel` znamená veľkosť jadra, v tomto prípade je to opäť veľkosť 5×5 pixelov. Funkcia `cv2.dilate()` nám zaistí to, že spojí biele pixely, ktoré sa nachádzajú blízko pri sebe, ale sú oddelené. Funguje to tak, že sa všetky pixely obrázku prekrývajú kernelom a ak je aspoň jeden pixel biely, všetky pixely pod kernelom budú biele. Argumentmi funkcie



Obr. 5.2: **Funkcia Houghovej transformácie**

sú čierno-biely obrázok, v tomto prípade ten, ktorý sme dostali v predchádzajúcom kroku, ďalej jadro, je použitá matica o veľkosti 5×5 naplnená jednotkami a posledný argument je počet iterácií. Druhou funkciou je `cv2.erode()`. Tá funguje presne opačne ako funkcia `cv2.dilate()`. Prechádza všetky pixely vstupného obrázku jadrom a ak nie sú všetky pixely pod jadrom biele, tak výsledný pixel bude čierny. Funkcii predáme výsledok predošlej funkcie, jadro a počet iterácií. Často sa tieto funkcie používajú v opačnom poradí, ale v tomto prípade je potrebné takéto poradie a to z dôvodu, že niektoré dokumenty budú mať ohnuté strany, teda ich nezachytí funkcia `cv2.HoughLinesP()`. Taktiež môžu byť okraje týchto strán nedostatočne detekované pomocou `cv2.Canny()`, preto je potrebné jednotlivé body najskôr zvýrazniť a následne ich zmenšiť, aby sa vo výslednom obrázku dokumentu nenachádzalo jeho okolie. Týmto sme sa dostali na koniec funkcie `preProcessing()`. Výsledný čierno-biely obrázok vrátime pomocou `return treshImg` a pokračujeme v programe.

Detekcia kontúr

Ďalšou časťou je nájdenie najväčšieho tvaru, v tomto prípade štvorca alebo obdĺžnika. Funkcii `getContours()` predáme ako jediný argument obrázok `threshImg`, ktorý sme dostali pomocou funkcie `preProcessing()`. Na začiatku funkcie sú zadeklarované potrebné pomocné premenné:

```
biggest = numpy.array([])
maxArea = 0
```

Premenná `biggest` je prázdne pole, do ktorého sa uložia koordináty najväčšieho objektu detekovaného na obrázku a premenná `maxArea` slúži na zapamätanie plochy posledného najväčšieho objektu.

```
contours, hierarchy = cv2.findContours(img, cv2.RETR_EXTERNAL,
    cv2.CHAIN_APPROX_NONE)
```

Funkcia `cv2.findContours()` dokáže nájsť kontúry v binárnom obraze. Kontúry sú body, ktoré určujú hranice tvarov. Funkcia vracia kontúry a ich hierarchiu. Všetky deteko-

vané objekty majú nejakú hierarchiu. Niekedy sú objekty umiestnené v rozličných miestach, niekedy je jeden objekt v inom. V tomto prípade je vonkajší objekt rodičovský a vnútorný objekt synovský. To znamená, že kontúry majú medzi sebou určité vzťahy. Na základe toho môžme určiť, ktoré objekty majú medzi sebou nejaký súvis a ktoré sú rodičovské a synovské. Prvým argumentom funkcie je vstupný obrázok, druhým je tzv. Contour Retrieval Mode (režim načítania kontúr), čo je vlastne spôsob, aké kontúry funkcia vráti. V tomto prípade je to `cv2.RETR_EXTERNAL`, čo znamená, že funkcia vráti len vonkajšie rodičovské objekty, všetky objekty nachádzajúce sa v ich vnútri nebudú brané do úvahy. Tretím parametrom je metóda aproximácie obrysu. `cv2.CHAIN_APPROX_NONE` znamená, že budú uložené všetky body obrysu. Vo výstupnej premennej `contours` sa bude po vykonaní funkcie nachádzať zoznam, ktorého položky tvoria polia. Ďalej sa pozrieme na blok kódu, ktorý nám odhalí najväčší objekt v obraze:

```
try:
    for contour in contours:
        area = cv2.contourArea(contour)
        if area > (self.height * self.width / 3):
            peri = cv2.arcLength(contour, True)
            approx = cv2.approxPolyDP(contour, 0.02 * peri, True)
            if area > maxArea and len(approx) == 4:
                biggest = approx
                maxArea = area

except Exception:
    biggest = []
```

Na začiatku je použitý cyklus `for`, pomocou ktorého je prechádzaný zoznam `contours` a v premennej `contour` sa nachádzajú jednotlivé polia zoznamu. Pomocou funkcie `cv2.contourArea()` sa vypočíta plocha vo vnútri jednotlivých kontúr. V ďalšom kroku sa overuje, či je veľkosť plochy väčšia ako minimálne tretina vstupného obrázku. To sa sem nachádza z dôvodu, aby sa nespracovávali veľmi malé objekty, ktoré vieme, že nie sú dokumenty. To je jednou s podmienok správneho fungovania programu - dokumenty na fotografii musia byť dostatočne veľké, v opačnom prípade nebude mať výstupná snímka dokumentu dostatočnú kvalitu. Ďalšia použitá funkcia je `cv2.arcLenngth()`. Funkcia počíta dĺžku krivky alebo uzavretý obvod kontúry. Prvým argumentom je pole, v ktorom sa nachádzajú kontúry a druhým argumentom je boolovská premenná, ktorá ak je pravdivá, tak znamená, že sú kontúry uzavreté. Výstup tejto funkcie sa použije v funkcii `cv2.approxPolyDP()`, ktorá aproximuje tvar obrysu inému tvaru s menším počtom vrcholov v závislosti od zadanej presnosti. To znamená, že ak chceme v obraze nájsť štvorec, ale kvôli rôznym problémom v obraze sme nezískali "perfektný" štvorec, ale napríklad skosený štvorec. Táto funkcia sa používa na aproximáciu tohoto tvaru. Prvým argumentom je pole s kontúrami, druhým argumentom je vzdialenosť kontúry od približnej kontúry. Určuje sa tým presnosť, v tomto prípade 0.02-násobok výsledku predošlej funkcie. Tretím argumentom je rovnako ako v predchádzajúcej funkcii boolovská premenná, ktorá ak je pravdivá, tak znamená, že sú kontúry uzavreté. Vďaka tejto funkcii si overíme, či sa naozaj jedná o štvorec/obdĺžnik. Následne si overíme, či plocha práve spracovávaného objektu je väčšia ako plocha objektu, ktorý bol doteraz najväčší. Taktiež sa overí, že objekt má 4 vrcholy - `len(approx) == 4`. Ak sú obe podmienky splnené, spracovávaný objekt je určený za najväčší, uloží sa do premennej `biggest` a veľkosť jeho obsahu sa uloží do premennej `maxArea`. Takto sa v cykle skontrolujú

všetky objekty v zozname `contours` a vráti sa ten najväčší štvorec/obdĺžnik. V prípade, že by bol zoznam `contours` prázdny, alebo by sa v zozname nenašiel žiaden dostatočne veľký štvorec, tak bude premenná `biggest` typu pole prázdna. Na konci funkcie `getContours()` sa vráti pole `biggest`.

Pokračujeme vo funkcii `run()`. Overíme si, či sa v premennej `biggestImg`, ktorú vrátila funkcia `getContours()` niečo nachádza. Ak áno, pokračujeme ďalej, ale ak je premenná prázdna, musíme nejakým spôsobom získať okraje dokumentu, ktorý sa nachádza na snímke, ale doteraz použitými metódami sa ho nepodarilo automaticky detekovať, alebo ho užívateľ chce označiť manuálne. Na to si vytvoríme inštanciu triedy AnotherWindow.

Trieda AnotherWindow

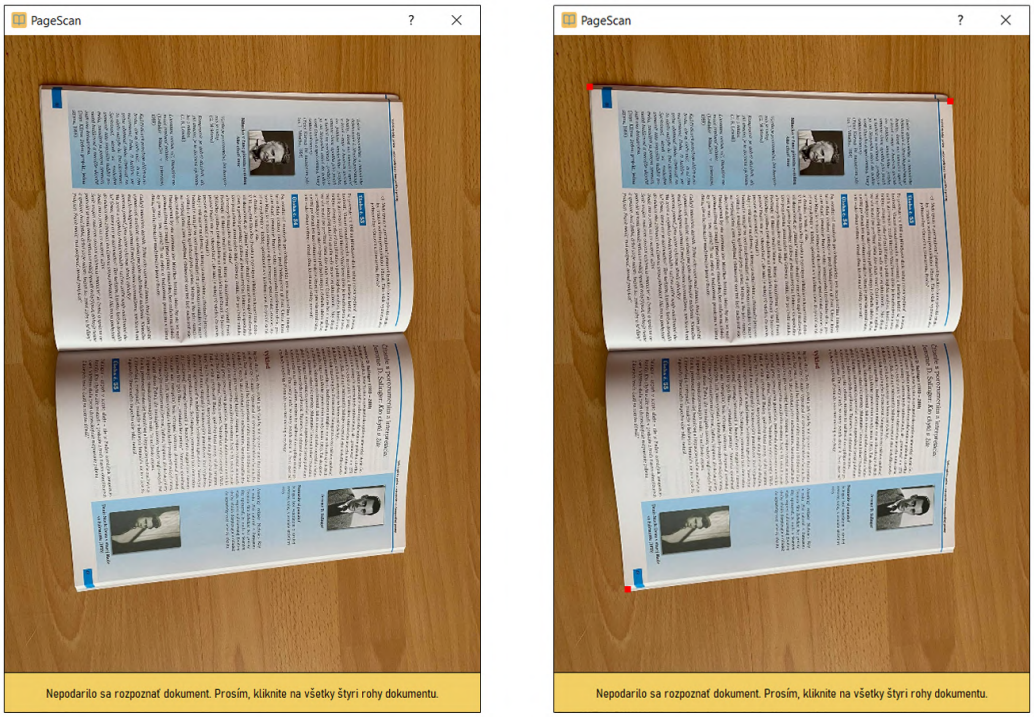
Trieda `AnotherWindow` sa taktiež nachádza v súbore `program.py`. Využíva knižnicu PyQT, o ktorú sa budeme viac zaujímať v ďalšej kapitole. Podstatné je, že sa vytvorí nové okno typu `QDialog`. Trieda obsahuje premennú `array`, ktorá je inicializovaná ako prázdne pole, premennú `idx`, ktorá sa rovná nule, premennú `_want_to_close`, ktorá sa rovná `False` a taktiež pole štyroch položiek typu `QLabel` s názvom `points`, ktoré majú veľkosť 8×8 pixelov. Vytvorené okno má rozmery 600 pixelov na šírku a 850 pixelov na výšku. Okno je rozdelené na 2 časti, horná časť má 800 pixelov na výšku a spodná časť 50 pixelov na výšku a obe časti zaberajú celú šírku vytvoreného okna. V hornej časti sa nachádza snímka, s ktorou sa v programe momentálne pracuje. V dolnej časti sa nachádza informácia, prečo sa zobrazilo toto okno a čo je treba spraviť. Informácia znie: `"Nepodarilo sa rozpoznať dokument. Prosím, kliknite na všetky štyri rohy dokumentu."`. Od užívateľa sa očakáva, že označí okraje dokumentu na snímke. Ak sa užívateľ pokúsi zavrieť okno kliknutím na tlačidlo X v pravom hornom rohu okna, tak sa mu to nepodarí, pretože udalosť bude odignorovaná pomocou funkcie `closeEvent()`. Samozrejme, že sú aj iné spôsoby, ako zavrieť toto okno. Ak sa však zavrie, program sa preruší a dokument nebude spracovaný. Ak však užívateľ počúvne informáciu v spodnej časti okna, a klikne na okraje dokumentu, pri každom kliku sa spustí funkcia `mousePressEvent()`. Najskôr sa overí, či užívateľ klikol na oblasť, v ktorej sa nachádza snímka, teda v hornej časti. Ak nie, tak sa kliknutie odignoruje. Ak však klikol na snímku, pri každom kliku sa na miesto kliknutia vykreslí jeden zo štyroch prvkov uložených v poli `points` tak, že sa presunie na miesto, kde užívateľ klikol a zmení farbu svojho pozadia na červenú. To slúži ako spätná väzba pre užívateľa. Následne sa zvýši hodnota premennej `idx` o jednu. Do poľa `array` sa uložia koordináty x a y kliknutia myši. Príklad zobrazenia okna je možné vidieť na obrázku 5.3. Na konci funkcie `mousePressEvent()` sa kontroluje, či je v poli `array` 8 položiek. Ak áno, okno sa zatvorí, do premennej `biggestImg` vo funkcii `run()` sa uložia koordináty okrajov dokumentu na snímke a pokračuje sa funkciou run().

Pokračuje sa zavolaním funkcie `resizeBiggest()`:

```
biggestImg = self.resizeBiggest(biggestImg, original_height,
    original_width)
```

Predajú sa jej 3 argumenty: prvý je pole bodov z predchádzajúceho kroku, ďalším je pôvodná výška snímky a posledným pôvodná šírka snímky. Najskôr sa zistí pomer medzi pôvodnou výškou/šírkou a výškou/šírkou, s ktorou sme doteraz pracovali:

```
height_ratio = original_height / self.height
width_ratio = original_width / self.width
```



Obr. 5.3: **AnotherWindow**

Nasledne sa v cykle `for` prechádzajú všetky prvky poľa `biggestImg` a jednotlivé prvky poľa sa násobia premennými `height_ratio` a `width_ratio`.

```
for row in biggestImg:
    row[0][0] = row[0][0] * width_ratio
    row[0][1] = row[0][1] * height_ratio
```

Tento krok je dôležitý, pretože doteraz sme kvôli efektivite výpočtov pracovali so zmenšeným obrázkom, ale v ďalších krokoch budeme upravovať obrázok v pôvodnej veľkosti, aby sme predišli strate kvality. Na konci funkcie sa pomocou `return biggestImg` vráti upravené pole okrajov dokumentu.

Orezanie snímky

Ďalej sa vo funkcii `run()` zavolá funkcia `getWarp()` s dvoma argumentmi: prvý je pôvodný obrázok, ktorý je uložený v premennej `copy_img` a druhý je pole `biggestImg`.

```
warpedImg = self.getWarp(copy_img, biggestImg)
```

Na začiatku funkcie `getWarp()` sa vytvorí prázdne pole `ret_page` a zavolá sa funkcia `reorder()` s argumentom `biggest`, čo je pole `biggestImg` predané ako argument funkcii `getWarp()`. Keďže v tomto poli sú uložené koordináty okrajov dokumentu neusporiadané, je potrebné ich usporiadať tak, že prvý prvok je ľavý horný okraj dokumentu, druhý prvok je pravý horný okraj, tretí prvok ľavý dolný okraj a posledný štvrtý prvok je pravý dolný okraj dokumentu. Najskôr je však potrebná zmena poľa, v ktorom sú uložené okraje dokumentu. To sa vykoná funkciou `numpy.reshape()` s dvoma argumentmi: prvým je pole

`points` a druhým je nový tvar poľa, v tomto prípade je to (4, 2), čo značí pole so štyrmi riadkami a dvomi stĺpcami. Táto funkcia dokáže zmeniť tvar poľa bez zmeny jeho obsahu. Výsledkom funkcie sa prepíše premenná `points`. Ďalej sa vytvorí prázdne pole `newPoints`, do ktorého sa budú ukladať usporiadané koordináty. do premennej `add` si pomocou funkcie `points.sum(1)` uložíme súčet prvkov poľa `points`, takže nám vznikne pole o štyroch prvkoch, kde každý prvok bude súčtom jedného stĺpca poľa `points`. Následne sa zistí ľavý horný a pravý dolný okraj dokumentu, čiže prvý a posledný prvok poľa `newPints`.

```
newPoints[0] = points[numpy.argmin(add)]
newPoints[3] = points[numpy.argmax(add)]
```

Pomocou funkcie `numpy.argmin(add)` sa zistí index prvku poľa `add`, ktorý má najmenšiu hodnotu. Logicky vyplýva, že najmenšiu hodnotu bude mať prvok s koordinátmi najbližšími k počiatku súradnicového systému, teda k ľavému hornému rohu snímky. Naopak, funkciou `numpy.argmax(add)` sa zistí index prvku poľa `add` s najväčšou hodnotou. Taktiež sa dá jasne určiť, že je to pravý dolný okraj dokumentu, pretože je najďalej od počiatku súradnicového systému. Následne si vypočítame rozdiel medzi prvkami poľa `points` a uložíme ich do poľa `diff`. Rozdiel sa vypočíta ako `out[i] = in[i+1] - in[i]`. Keďže v tomto prípade je `in[i+1]` hodnota na ose y a `in[i]` hodnota na ose x, tak z toho vyplýva, že najmenšiu hodnotu bude mať prvok poľa obsahujúci koordináty bodu nachádzajúci sa vpravo hore, keďže rozdiel hodnôt jeho osí y a x bude určite menší ako rozdiel hodnôt osí y a x ostatných prvkov. Podobne, rozdiel hodnôt osí y a x prvku s koordinátom, ktorý ukazuje ľavý dolný okraj dokumentu bude určite väčší, ako rozdiel hodnôt osí y a x ostatných prvkov.

```
diff = numpy.diff(points, axis=1)
newPoints[1] = points[numpy.argmin(diff)]
newPoints[2] = points[numpy.argmax(diff)]
```

Môžme tvrdiť, že v poli `newPoints` sa nachádzajú zoradené koordináty okrajov dokumentu na snímke. Toto pole vrátime a pokračujeme vo funkcii `getWarp()`. Výsledok funkcie `reorder()` sa uloží do poľa `biggest` a prepíšu sa tak pôvodné hodnoty. Ďalej sa vytvorí pole `points1`, čo je vlastne kópia poľa `biggest`, ale s hodnotami typu `float32`. Nasleduje kontrola, či je na snímke jednostranný alebo dvojstranný dokument. To sa určí na základe premennej `two_pages`. Najskôr sa budeme zaoberať jednostranným dokumentom. Overí sa, či je šírka strany väčšia ako jej výška. To sa môže stať v prípade, že je na vstupe dvojstranný dokument, ale užívateľ chce manuálne detekovať len jednu stranu. Kvôli tomu je nutné preusporiadať pole, v ktorom sa nachádzajú koordináty okrajov dokumentu. Následne sa vypočíta výška a šírka výsledného dokumentu. Výška sa zistí z rozdielu súradníc ľavého dolného a ľavého horného okraja dokumentu. Šírka sa následne vypočíta vynásobením výšky hodnotou 0,707. Táto hodnota označuje pomer výšky a šírky dokumentu A4. Potom sa vykoná nasledujúca akcia:

```
points1[0][0][0] += int(self.top * self.a4_height * 0.02)
points1[1][0][0] += int(self.top * self.a4_height * 0.02)
points1[2][0][0] -= int(self.bottom * self.a4_height * 0.02)
points1[3][0][0] -= int(self.bottom * self.a4_height * 0.02)
```

Keďže máme správne označené okraje dokumentu, v prípade, že by bola strana ohnutá, došlo by k orezaniu jej hornej a dolnej časti. Môžme si predstaviť ohnutú stranu, naniesť si priamku spájajúcu ľavý horný roh strany s pravým horným rohom strany. Všetko čo by sa nachádzalo nad priamkou, by bolo orezané a to nechceme. Preto sa horné okraje posunú

vyššie a dolné okraje posunú nižšie v závislosti od veľkosti snímky a veľkosti zahnutia uloženého v premenných `top` a `bottom`. V prípade, že by bol na vstupnej snímke len jednostranný dokument, vykonajú sa podobné operácie, ale nemení sa usporiadanie okrajov dokumentu.

Ďalším krokom je vytvorenie poľa, do ktorého bude uložený výsledný dokument. Pole bude mať šírku v závislosti na hodnote uloženej v premennej `a4_width` a výšku o veľkosti hodnoty v premennej `a4_height`. Pomocou funkcie `cv2.getPerspectiveTransform()` sa vytvorí transformačná matica o veľkosti 3×3 s názvom `matrix`.

```
matrix = cv2.getPerspectiveTransform(points1, points2)
```

Táto funkcia vypočíta perspektívnu transformáciu medzi dvomi zadanými bodmi. Matica vznikne následovne:

$$\begin{bmatrix} t_i x'_i \\ t_i y'_i \\ t_i \end{bmatrix} = \text{matrix} \cdot \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

kde:

$$\text{points2}(i) = \left(x'_i, y'_i\right), \text{points1}(i) = (x_i, y_i), i = 0, 1, 2, 3$$

Prvý argument funkcie je vstupné pole, v tomto prípade `points1`, druhým parametrom je výstupné pole, `points2`. Ďalšia použitá funkcie je `cv2.warpPerspective()`.

```
page = cv2.warpPerspective(img, matrix, (self.a4_width, self.a4_height))
```

Táto funkcia aplikuje perspektívnu transformáciu na snímku. Prvým argumentom je snímka, druhým je transformačná matica a tretím je veľkosť výslednej snímky.

V ďalšej časti sa rozhoduje, či je strana zahnutá alebo nie. To sa rozhoduje na základe súčtu premenných `top` a `bottom`. V prípade, že sa ich súčet rovná nule, orezaná strana sa uloží a pokračuje sa vo funkcii `run()`. Ak je však ich súčet nenulový, pokračuje sa v rozhodovaní, či je strana ľavá alebo pravá. V prípade, že je na snímke jeden list papiera, tak na tom nezáleží, ale v prípade, že máme napríklad otvorenú knihu, ale vyfotená je len jedna strana, je potrebné určiť, či je táto strana ľavá alebo pravá. Je to dôležité z hľadiska ďalšieho spracovania a vyrovnania ohnutých strán. Kvôli správnej činnosti ďalších funkcií je potrebné snímku otočiť o 90° v smere hodinových ručičiek pomocou funkcie:

```
page = cv2.rotate(page, cv2.ROTATE_90_CLOCKWISE)
```

Na základe premennej `left_page` sa v bloku `if` rozhodne, či je strana ľavá alebo pravá a na základe toho sa zavolá funkcia `left()` pre ľavú stranu alebo `right()` pre pravú stranu. Najskôr sa pozrieme na funkciu `left()`.

Na začiatku funkcie si zistíme rozmery snímky:

```
hp, wp, chp = page.shape
```

, kde `hp` je výška snímky, `wp` je šírka snímky a `chp` je počet kanálov, ale s tým nebudeme pracovať. Následne pomocou výpočtu

```
mid = int(self.bottom / (self.top + self.bottom) * wp)
```

získame hranicu snímky, ktorá nám určuje, pokiaľ budú aplikované ďalšie algoritmy. Je to vlastne pomer zahnutia hornej a dolnej časti strany. V tomto mieste by mala byť snímka bez zahnutia. Ďalej sa vytvoria dva pomocné polia, do jedného sa bude ukladať spracovaná časť snímky pod hranicou `mid` a do druhého časť snímky nad hranicou `mid`.

```
result1 = numpy.zeros((self.a4_width, mid, 3), numpy.uint8)
result2 = numpy.zeros((self.a4_width, wp - mid, 3),
    numpy.uint8) numpy.uint8)
```

Následne sa snímka rozdelí na dve časti a obe časti sa uložia do polí `page1` a `page2`.

```
page1 = page[0: self.a4_width, 0: mid]
page2 = page[0: self.a4_width, mid: wp]
```

Kvôli správnemu výpočtu je spodnú časť strany, teda pole `page1`, potrebné obrátiť o 180° pomocou funkcie `cv2.rotate()`. Ďalšou dôležitou časťou je výpočet kroku, o ktorý sa bude posúvať v poli `bend_arr`.

```
step = self.a4_width / len(self.bend_arr)
```

Nasleduje cyklus for, v ktorom sa spracúvajú všetky riadky snímky. Netreba zabudnúť, že snímka dokumentu je momentálne otočená o 90° doprava, čiže sa vlastne spracúvajú stĺpce dokumentu. Práve kvôli tomu, ako sa cykle pracuje s poľami, bolo potrebné snímku otočiť. V cykle sa rieši, či sú premenné `bottom` a `top` väčšie ako 0, ak nie sú, nie je potrebné s konkrétnou časťou snímky vykonávať žiadnu akciu. Zameriame sa na spodnú časť snímky:

```
adj_width = self.bottom * self.a4_height * 0.02 *
    self.bend_arr[len(self.bend_arr) - 1 - int(row // step)]
points1 = numpy.float32([[row, 0], [row, mid - self.bottom *
    self.a4_height * 0.02 + adj_width], [row + 1, 0],
    [row + 1, mid - self.bottom * self.a4_height * 0.02 + adj_width]])
points2 = numpy.float32([[row, 0], [row, mid], [row + 1, 0],
    [row + 1, mid]])
matrix = cv2.getPerspectiveTransform(points1, points2)
result1[row] = cv2.warpPerspective(page1[row], matrix, (3, mid))
```

Premenná `adj_width` znamená, o koľko pixelov je zväčšený konkrétny riadok matice, v tomto prípade matice `page1`. Dá sa to chápať ako korekcia zahnutia strán. Ďalej sa zadá pole `points1`, ktoré určuje šírku každého riadku. V ďalšom kroku je zadané pole `points2`. To určuje rozmery a umiestnenie výsledného poľa. Pomocou funkcie `cv2.getPerspectiveTransform()` sa vypočíta transformačná matica, ktorá je použitá vo funkcii `cv2.warpPerspective()`. Výsledok tejto funkcie uloží riadok časti snímky uloženej v poli `page1` do výsledného poľa `result1`. Toto bol výpočet pre spodnú časť strany. Výpočet hornej časti strany prebieha podobne, je tam len niekoľko zmien, napríklad pri výpočte dolnej časti strany sa pole `bend_arr` prechádzalo odzadu a to z dôvodu, že bola predtým táto časť otočená o 180°, zatiaľ čo pri výpočte hornej časti strany sa pole `bend_arr` prechádza od začiatku. Taktiež sa miesto premennej `bottom` používa premenná `top`. Celá táto časť vyzerá následovne:

```
adj_width = self.top * self.a4_height * 0.02 *
    self.bend_arr[int(row // step)]
points1 = numpy.float32([[row, 0], [row, wp - mid - self.top *
    self.a4_height * 0.02 + adj_width], [row + 1, 0],
    [row + 1, wp - mid - self.top * self.a4_height * 0.02 + adj_width]])
points2 = numpy.float32([[row, 0], [row, wp - mid], [row + 1, 0],
    [row + 1, wp - mid]])
matrix = cv2.getPerspectiveTransform(points1, points2)
result2[row] = cv2.warpPerspective(page2[row], matrix, (3, wp - mid))
```

Takto sa v cykle for prejdú všetky riadky snímky a pomocou perspektívnej transformácie sa minimalizujú zahnutia strán (samozrejme pri zvolení správnych hodnôt `top` a `bottom`). Keďže boli časti snímky otočené o 90°, je potrebné ich vrátiť do pôvodného stavu:

```
result1 = cv2.rotate(result1, cv2.ROTATE_90_CLOCKWISE)
result2 = cv2.rotate(result2, cv2.ROTATE_90_COUNTERCLOCKWISE)
```

V ďalšej časti je potrebné tieto oddelené časti spojiť. Najskôr je však potrebné zistiť, či sa upravovali obe časti, len vrchná časť, alebo len spodná časť:

```
if self.top > 0 and self.bottom > 0:
    result = numpy.concatenate((result2, result1), axis=0)
elif self.top == 0:
    result = result1
else:
    result = result2
```

V prípade, že boli upravované obe časti, je potrebné ich spojiť do jednej pomocou funkcie

```
result = numpy.concatenate((result2, result1), axis=0)
```

Výsledná snímka je vrátená funkcii `getWarp()` pomocou `return result`.

Funkcia `right()` je podobná funkcii `left()`, ale líši sa v cykle, kde sa počítajú nové rozmery každého riadku snímky. Po vrátení snímky funkcii `getWarp()` sa snímka uloží do poľa `ret_page`, ktoré sa na konci funkcie vráti funkcii `run()`.

V prípade, že je premenná `two_pages` rovná `True`, čiže máme snímku s dvomi stranami, postupujeme podobne ako pri jednej strane. Najskôr sa vypočíta výška a šírka výsledného dokumentu. Ďalej sa snímka oreže s dodatočnými okrajmi kvôli ohnutým stranám. Tentokrát však inak ako pri jednostrannom dokumente. Tam sme pridávali pixely na ose y, tentokrát musíme pridať pixely na ose x, pretože snímka dvoch strán je otočená o 90° doprava.

```
points1[0][0][0] -= int(self.bottom * self.a4_height * 0.02)
points1[1][0][0] += int(self.top * self.a4_height * 0.02)
points1[2][0][0] -= int(self.bottom * self.a4_height * 0.02)
points1[3][0][0] += int(self.top * self.a4_height * 0.02)
```

Po určení okrajových bodov je potrebné určiť výsledné pole, do ktorého bude snímka pomocou perspektívnej transformácie uložená. Tentokrát je výsledná matica logicky dvakrát väčšia ako matica pri jednostrannom dokumente. Vypočíta sa transformačná matica a opäť sa použije funkcia `cv2.warpPerspective()`:

```
points2 = numpy.float32([[0, 0], [self.a4_height, 0],
    [0, 2 * self.a4_width], [self.a4_height, 2 * self.a4_width]])
matrix = cv2.getPerspectiveTransform(points1, points2)
page = cv2.warpPerspective(img, matrix, (self.a4_height,
    2 * self.a4_width))
```

Keď sme získali orezanú snímku, je potrebné ju rozdeliť na ľavú a pravú časť. Existujú rôzne spôsoby orezania, môže sa opäť použiť detekcia okrajov strán, ale počas testovania to nefungovalo dostatočne správne, pretože vo veľkej časti snímok nebolo možné detekovať miesto, kde sa strany spájajú. Preto je použité orezanie s pevnou veľkosťou, ale vždy je na strane spojenia strany pridaných pár pixelov, aby sa náhodou neorezala časť strany. Počas testovania to splnilo svoj účel.

```
page1 = page[0 : self.a4_width + int(self.a4_width * 0.01),
    0 : self.a4_height]
page2 = page[self.a4_width - int(self.a4_width * 0.01):
    2 * self.a4_width, 0: self.a4_height]
```

Následne sa podobne ako pri jednostrannom dokumente zavolajú funkcie `left()` pre ľavú stranu a `right()` pre pravú stranu. Obe strany sú pridané do zoznamu `ret_page`, ktorý je vrátený funkcii `run()`. V ďalšej časti sa zavolá funkcia `change_colors()`. Na základe obsahu premennej `colors` sa buď nevykoná žiadna zmena, alebo sa farby snímok zmenia do odtieňov šedej, alebo len do čiernobielej. Zmena farieb do čiernobielej sa vykonáva pomocou funkcie `cv2.adaptiveThreshold()`, ktorá získa výslednú farbu pixelu na základe jeho okolia. Veľkosť okolia sa vypočíta na základe veľkosti snímky a uloží sa do premennej `block_size`. Poslednou možnosťou úpravy farby je zmena kontrastu. Existujú 4 spôsoby zmeny kontrastu: Zvýšenie o 20% alebo o 40% a zníženie o 20% alebo o 40%. To sa vykonáva zavolaním funkcie `contrast()` s argumentom zvýšenia/zníženia kontrastu. Tu sa využijú funkcie `Contrast()` a `enhance()` knižnice `Pillow`. Snímka so zmeneným kontrastom sa vráti funkcii `change_colors()`. Tá nakoniec vráti zoznam obrázkov funkcii `run()`.

Na záver sa zmení aktuálny pracovný adresár na adresár, ktorý bol určený ako výstupný. Potom sa v cykle spracujú snímky zo zoznamu `warpedImg`. Získa sa meno a prípona každého súboru. Overí sa, či sa vo výstupnom priečinku nenachádza súbor s rovnakým názvom, ak áno, zmení sa jeho názov pridaním čísla. Následne sa súbor uloží do výstupného priečinka. Posledným krokom je možnosť zobrazenia výstupného súboru v prípade, že je premenná `show_files` rovná `True`.

5.2 Frontend časť

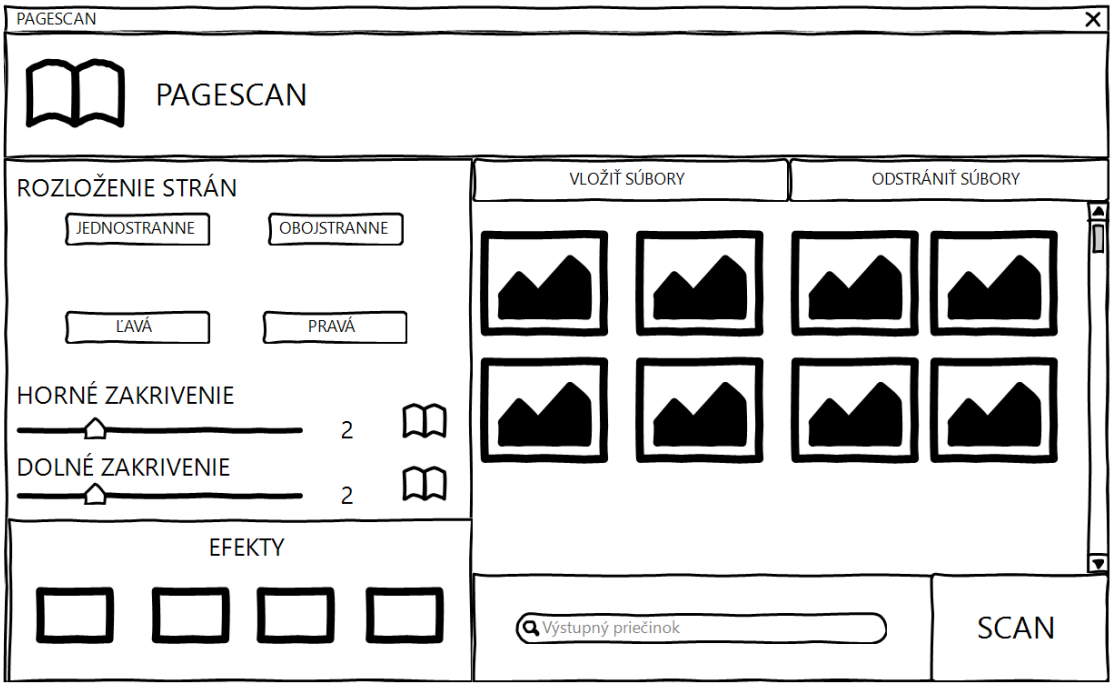
Táto časť programu sa venuje vizuálnej stránke aplikácie a je uložená v súbore `app.py`. Zdrojom informácií pre túto časť boli stránky dokumentácie[2] pre Qt. Ešte predtým, ako sa začalo programovať, bolo potrebné vytvoriť grafický návrh. Ten musel vychádzať z požiadaviek, ktoré boli popísané v zadaní práce.

Grafický návrh

Prvým krokom grafického návrhu je zistiť, pre koho bude výsledný produkt určený. V tomto prípade sa naskytuje hneď niekoľko možností. Tou prvou sú bežní užívatelia, ktorí potrebujú upraviť zopár vyfotených dokumentov, prípadne zdigitalizovať knihu. Táto skupina užívateľov je veľmi široká a zahŕňa ľudí, ktorí nie sú veľmi zdatní v práci s počítačom, až po ľudí, ktorí im dokonale rozumejú. Ďalšou skupinou sú vedeckí pracovníci, ktorí potrebujú upraviť určité dokumenty, prípadne zdigitalizovať staršie záznamy. Podobne to je aj s ľuďmi, ktorí pracujú v knižniciach, alebo sa zaoberajú spracovávaním dokumentov, alebo kníh. Keď je už známe, pre koho bude program určený, je nutné zozbierať čo najviac informácií o potrebách užívateľov. Cieľom je zabezpečiť, aby boli všetky potrebné nastavenia logicky oddelené a aby bolo na prvý pohľad jasné, čo ktoré z nastavení robí. Z tohoto dôvodu je potrebné vytvoriť grafický návrh tak, aby sa v ňom rýchlo zorientoval čo najväčší počet ľudí, ktorí budú pracovať s týmto programom. Keďže podobných aplikácií veľa neexistuje, je nutné vytvoriť niečo nové, čo užívateľa zaujme, ale pritom je potrebné zabezpečiť to,

[2] https://doc.qt.io/

že sa užívateľ nebude strácať v jednotlivých možnostiach programu. Ďalšou možnosťou je kontaktovať ľudí z cieľovej skupiny užívateľov a pravidelne od nich získavať spätnú väzbu v procese tvorby návrhu. Na začiatok je najlepšie vytvoriť niekoľko nákresov, vybrať tie najlepšie a postupne ich zdokonaľovať. Takto to prebiehalo aj v tomto prípade. Prvotný návrh je možné nakresliť na papier, ale lepšie je jeho realizácia pomocou nejakého nástroja, ako je napríklad editor `WireframeSketcher`. Výsledný návrh rozmiestnenia prvkov je vidieť na obrázku 5.4

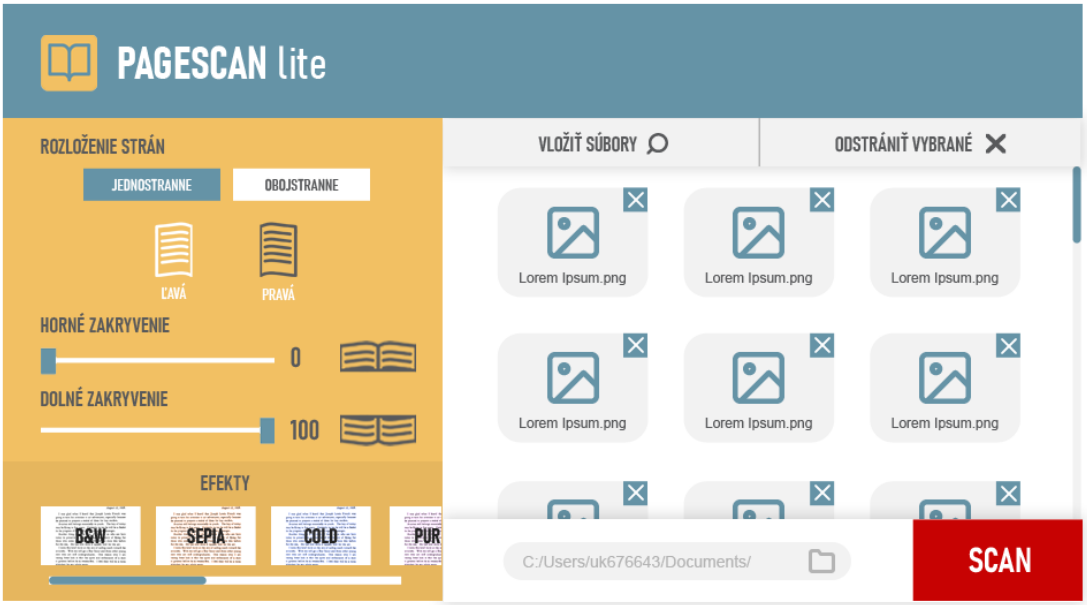


Obr. 5.4: **Mockup vytvorený v programe WireframeSketcher**

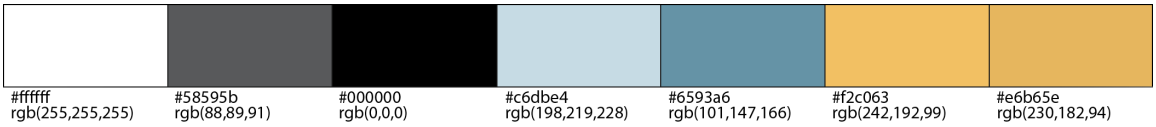
Po vytvorení prvotného návrhu je nutné ho prerobiť do grafického návrhu, ktorý bude obsahovať konkrétne farby, písma, rôzne detaily. Na to bol použitý program `Adobe Photoshop`. V tomto nástroji sa dajú vytvoriť akékoľvek návrhy, vložiť texty rôznych typov, farieb, je možné pohrať sa s farbami a rôznymi detailami. Po rôznych úpravách vznikol prvotný grafický návrh finálnej aplikácie. Tento návrh je možné vidieť na obrázku 5.5

V návrhu je použitý font `Bahnschrift`. Farby, ktoré boli použité pri návrhu je možné vidieť na obrázku 5.6

V hornej časti návrhu sa na celú šírku rozprestiera modrý blok a v jeho ľavej časti sa nachádza logo aplikácie a taktiež jej predbežný názov - PAGESCAN lite. Aplikácia je následne rozdelená na dve časti. Ľavá časť slúži predovšetkým na nastavenie parametrov, zatiaľ čo pravá strana slúži na správu vstupných súborov a umiestnenie výstupného priečinku. Teraz si bližšie rozoberieme ľavú časť. V hornej časti sa nachádza nastavenie, či pracujeme s jednostranným, alebo dvojstranným dokumentom. V prípade, že je vybratá možnosť JEDNOSTRANNE, je možnosť výberu, či je táto strana ľavá alebo pravá. Po označení možnosti OBOJSTRANNE sa výber ľavej alebo pravej strany skryje. Hneď pod tým sa nachádza možnosť výberu veľkosti horného a dolného zakrivenia. Je to realizované pomocou posuvníku, vedľa neho sa nachádza jeho hodnota a napravo sa nachádza ikona, ktorá má za úlohu lepšie vysvetliť užívateľovi konkrétne nastavenie. V spodnej časti sa nachádza výber z možných



Obr. 5.5: **Návrh vo Photoshope**



Obr. 5.6: **Vzorník farieb**

efektov, ktoré je možné aplikovať na snímky, či už to bude možnosť čierno-bieleho obrázku, obrázku v stupňoch šedej, alebo možnosť zvýšenia kontrastu. V pravej polovici grafického návrhu aplikácie je dominantná oblasť, v ktorej sa budú nachádzať snímky určené na spracovanie. Nad touto plochou sa nachádzajú dve tlačidlá. Jedno slúži na vkladanie súborov. Po jeho kliknutí sa zobrazí okno, v ktorom sa vyberú vstupné súbory. Druhé tlačidlo slúži na odstránenie snímok, ktoré užívateľ nechce upravovať. V dolnej časti sa nachádza riadok, do ktorého bude potrebné zadať cestu k výstupnému priečinku. Do tohto priečinku sa budú ukladať spracované snímky. V pravom dolnom rohu sa nachádza výrazné tlačidlo, ktorým sa spúšťa program. Po tom, ako bol dokončený návrh dizajnu aplikácie, je potrebné jeho naprogramovanie.

Táto časť programu sa nachádza v súbore app.py. Na začiatku sú naimportované všetky potrebné knižnice: `sys`, regex (`re`), `os`, `time`, potrebné časti knižnice `PyQt5`, ako `QtCore`, `QtGui` a `QtWidgets`. Taktiež sú tu naimportované súbory `program.py` a `help_widget.py`, o ktorý sa budeme zaujímať neskôr. Ďalej sú vytvorené premenné, v ktorých sa nachádza vzhľad prvkov, ktoré sa často menia, konkrétne:

- **QListWidget_full**: vzhľad okna, v ktorom sa nachádzajú vložené súbory, v prípade, že sa v ňom nachádzajú nejaké súbory,
- **QListWidget_empty**: vzhľad okna, v ktorom sa nachádzajú vložené súbory, v prípade, že je prázdne,

- **run_button_off**: vzhľad tlačidla na spustenie programu, keď je zablokované,
- **run_button_on**: vzhľad tlačidla na spustenie programu, keď je povolené.

Úplne v spodnej časti súboru sa nachádza hlavná funkcia, ktorá spúšťa celý program:

```
if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = MainWindow()
    MainWindow.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

Podmienka `if` v tomto prípade zabezpečí to, že sa program spustí, len ak ide o hlavný program, teda ak by bol daný súbor importovaný v inom programe, tak sa funkcia nespustí. Na začiatku sa vytvorí inštancia `QApplication()`. Následne sa vytvorí inštancia triedy `MainWindow` typu `QMainWindow`. Po jej vytvorení sa zavolá funkcia `setupUi()`, ktorá patrí tejto triede a predá sa jej objekt `MainWindow`. Táto funkcia slúži na Vytvorenie všetkých objektov v okne `MainWindow`. Na jej začiatku sa vytvoria potrebné premenné:

- **scanner**: do premennej sa uloží objekt vytvorený z triedy Program zo súboru program.py,
- **two_pages**: premenná udáva, či pracujeme s jednostranným alebo dvojstranovým dokumentom, implicitne je hodnota False,
- **left_page**: premenná udáva, či pracujeme s ľavou alebo pravou stranou, v prípade, že je premenná two_pages rovná False, implicitne je hodnota True,
- **top**: premenná udáva veľkosť zahnutia hornej strany, implicitne 0,
- **bottom**: premenná udáva veľkosť zahnutia dolnej strany, implicitne 0.

Následne sa oknu priradí názov a určia sa jeho rozmery. Veľkosť okna je pevne stanovená na šírku 1000 pixelov a výšku 710 pixelov. Veľkosť okna sa nedá meniť.

```
MainWindow.setObjectName("MainWindow")
MainWindow.setEnabled(True)
MainWindow.resize(1000, 710)
MainWindow.setMinimumSize(QtCore.QSize(1000, 710))
MainWindow.setMaximumSize(QtCore.QSize(1000, 710))
```

Ďalej je oknu nastavená ikona. Ikona je v tomto prípade logo aplikácie.

```
icon = QtGui.QIcon()
icon.addPixmap(QtGui.QPixmap("img/logo.png"),
    QtGui.QIcon.Normal, QtGui.QIcon.Off)
MainWindow.setWindowIcon(icon)
```

Ďalej sa vytvorí objekt `QWidget` a uloží sa do premennej `centralwidget`. Do tohto widgetu sa budú vkladať všetky ďalšie prvky okna.

```
self.centralwidget = QtWidgets.QWidget(MainWindow)
self.centralwidget.setObjectName("centralwidget")
```

Prvky, ktoré sa budú vkladať do tohto widgetu sa rozdelia na 5 častí:

- **top_frame**: bude obsahovať hornú časť s logom a názvom aplikácie,
- **left_frame**: bude obsahovať nastavenia počtu strán a zakrivenia ich okrajov,
- **bottom_left_frame**: bude obsahovať nastavenia farieb
- **right_frame**: bude obsahovať oblasť, do ktorej sa budú vkladať súbory a tlačidlá na operácie s nimi,
- **bottom_right_frame**: bude obsahovať možnosť výberu výstupného priečinku a tlačidlo na spustenie programu.

top_frame

`top_frame` má šírku okna a výšku 121 pxielov a farbu rgb(101, 147, 166). Ďalej je vytvorené logo a to pomocou widgetu `QLabel`, ktorý má šírku 91 pixelov a výšku 81 pixelov. Pomocou nastavenia štýlov je doň vložený obrázok loga.

```
self.logo_img.setStyleSheet("border-image: url(img/logo.png)
    0 0 0 0 stretch stretch;")
```

Ďalším prvkom je názov aplikácie. Opäť je to widget `QLabel`. Poslednou časťou v časti `top_frame` je tlačidlo, ktoré po kliknutí naň zobrazí jednoduchý návod, na čo je program určený, ako sa sním pracuje, čo znamenajú jednotlivé prvky a čo sa s nimi nastavuje. Najskôr je vytvorený widget `QDialog`. Potom je vytvorený objekt `help_window`, v ktorom je uložená inštancia triedy `Ui_Dialog`.

```
self.dialog = QtWidgets.QDialog()
self.help_window = help_widget.Ui_Dialog()
self.help_window.setupUi(self.dialog)
```

Táto trieda sa nachádza v súbore `help_widget.py`. Opäť sa v spodnej časti dokumentu nachádza kód na spustenie programu:

```
if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    Dialog = QtWidgets.QDialog()
    ui = Ui_Dialog()
    ui.setupUi(Dialog)
    Dialog.show()
    sys.exit(app.exec_())
```

Tento kód sa však spúšťa, len ak sa spustí program v `help_widget.py` samostatne. Po zavolaní funkcie `self.help_window.setupUi(self.dialog)` v súbore `app.py` sa spustí funkcia `setupUi` v súbore `help_widget.py` a predá sa jej widget uložený v premennej `dialog` vytvorený krok predtým. Následne sa nastavia rozmery okna, je to 582 pixelov na šírku a 834 pixelov na výšku. Oknu sa taktiež nedá zmeniť veľkosť. Podobne ako pre `MainWindow`, je aj tu nastavená ikona okna. V ďalších krokoch sa vytvoria všetky widgety typu `QLabel`, ktorým sa nastaví veľkosť, pozícia a font, veľkosť, typ a zarovnanie písma. Na konci funkcie `setupUi()` sa zavolá funkcia `retranslateUi()`, ktorá do vytvorených objektov vloží text.

Po vytvorení a nastavení okna sa vraciame do súboru `app.py`. V ňom sa ďalej nastavuje vzhľad tlačidla na zobrazenie návodu. Je nastavená jeho veľkosť a pozícia a je mu priradená ikona v tvare otáznika.

```
self.help_button.clicked.connect(self.show_help)
```

Pomocou tejto funkcie je v prípade kliknutia na toto tlačidlo zavolaná funkcia `show_help()`, ktorá jednoducho zobrazí toto okno uložené v premennej `dialog`:

```
show_help(self):
    self.dialog.show()
```

left_frame

Ďalšou časťou je `left_frame`. Tomuto widgetu je najskôr nastavená veľkosť a pozícia. Má šírku 461 pixelov a výšku 421 pixelov, nachádza sa úplne vľavo okna `MainWindov` a zhora je posunutý na pozíciu 120 pixelov. Je mu nastavená farba rgb(242, 192, 99). Ďalej sa doň vloží nadpis označujúci rozloženie strán. Pod týmto nadpisom sa nachádzajú dva tlačidla, jedno s nápisom JEDNOSTRANNE a druhé s nápisom OBOJSTRANNE. Tieto tlačidlá sú typu `QPushButton`. Pred ich vytvorením je potrebné vytvoriť objekt, ktorý bude zapuzdrovať tieto tlačidlá, keďže sa v aplikácii nachádza viac tlačidiel a je potrebné ich medzi sebou oddeliť. Je preto vytvorený objekt `QButtonGroup`:

```
self.button_group = QButtonGroup(self.left_frame)
```

Po jeho vytvorení sú vytvorené aj tlačidlá, ktoré nastavujú, či má dokument na snímke jednu, alebo dve strany. Ako obyčajne, je tlačidlám nastavená ich veľkosť a pozícia. Taktiež je im nastavená farba pozadia a farba písma, ktorá odlišuje stlačené tlačidlo od nestlačeného. Stlačené tlačidlo (v tomto prípade tlačidlo s nápisom JEDNOSTRANNE) má farbu pozadia bielu a farbu písma čiernu, druhé tlačidlo má farbu pozadia rgb(101, 147, 166) a farbu písma bielu. Ako je možné si všimnúť, je to inak ako pri návrhu na obrázku 5.5, pretože pri testovaní sa zistilo, že pôvodné farby stlačených a nestlačených tlačidiel neboli jednoznačné. Tlačidlu, ktoré vyberá jednu stranu (s nápisom JEDNOSTRANNE) je nastavené, že je zakliknuté pomocou funkcie:

```
self.one_page_button.setCheckable(True)
self.one_page_button.setChecked(True)
```

Druhé tlačidlo má nastavenú hodnotu `setChecked` na `False`. Prvému tlačidlu je pri kliknutí naň priradená funkcia `one_page()`

```
self.one_page_button.clicked.connect(self.one_page)
```

a druhému funkcia two_page()

```
self.two_page_button.clicked.connect(self.two_page)
```

Obe tlačidla majú nastavenú zmenu kurzoru myši pri prechode cez ne:

```
self.one_page_button.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
```

Funkcia `one_page()` nastaví štýly tlačidiel pre výber jednej, alebo dvoch strán tak, aby bolo vidno, že je aktívne to, ktoré vyberá jednu stranu. Taktiež táto funkcia vykoná nasledujúce operácie:

```
self.left_page_button.show()
self.right_page_button.show()
self.left_label.show()
self.right_label.show()
self.two_pages = False
```

Najskôr zobrazí tlačidlá `left_page_button` a `right_page_button`, o ktorých sa dozvieme v ďalšej časti. Taktiež zobrazí nápisy pod týmito tlačidlami. Nakoniec nastaví hodnotu premennej `two_pages` na `False`. Funkcia `two_page()` robí presný opak:

```
self.left_page_button.hide()
self.right_page_button.hide()
self.left_label.hide()
self.right_label.hide()
self.two_pages = True
```

Najskôr nastaví štýl tlačidiel pre výber jednej alebo dvoch strán presne naopak, ako funkcia `one_page()` a následne skryje tlačidlá `left_page_button` a `right_page_button` a tiež nápisy pod nimi.

Po zakliknutí tlačidla na výber jednej strany je potrebné vybrať, či je táto strana ľavá alebo pravá. Kvôli tomu sa vytvoria už vyššie spomínané tlačidlá `left_page_button` a `right_page_button`, ktoré sú typu `QPushButton`. Podobne ako pri prvých dvoch tlačidlách, aj tu je vytvorený objekt `QButtonGroup`, do ktorého budú tieto dva tlačidlá zabalené. Po vytvorení tlačidiel sa im ako obyčajne nastaví veľkosť a umiestnenie a tiež vzhľad. Nebudú však vyzerať ako obyčajné tlačidlá, ale budú mať len obrázok. Tlačidlo na výber ľavej strany bude mať vzhľad ľavej strany a tlačidlo na výber pravej strany zase vzhľad pravej strany. Taktiež sa budú odlišovať farbami: Zakliknuté tlačidlo má bielu ikonu a nezaškrtnuté tlačidlo má farbu rgb(88, 89, 91), tak ako aj všetky ostatné ikony v aplikácii. Zmena farby ikony sa vykonáva zmenou celej ikony - pre ikonu ľavej aj pravej strany existuje biela varianta a tmavá varianta. Ikony sa menia zmenou štýlu. Ľavé tlačidlo sa nastaví ako zaškrtnuté:

```
self.left_page_button.setCheckable(True)
self.left_page_button.setChecked(True)
```

Ľavému tlačidlu je priradená funkcia `set_left_page()`, ktorá sa spustí po kliknutí naň:

```
self.left_page_button.clicked.connect(self.set_left_page)
```

Táto funkcia zmení farbu tlačidiel ľavej a pravej strany a tiež nápisov pod nimi tak, aby bolo jasné, že je zakliknuté tlačidlo ľavej strany. Taktiež sa nastaví premenná `left_page` na hodnotu `True`.

Pravému tlačidlu je zasa priradená funkcia `set_right_page()`, ktorá sa spustí po kliknutí naň:

```
self.right_page_button.clicked.connect(self.set_right_page)
```

Funkcia `set_right_page()` robí presný opak toho, čo funkcia `set_left_page()`. Zmení farbu tlačidiel a nápisov, aby bolo jasné že je zaškrtnuté tlačidlo označujúce pravú stranu a premennú `left_page` nastaví na hodnotu `False`.

Po vytvorení tlačidiel sú vytvorené nápisy pod nimi. Oba nápisy sú typu `QLabel`, je im nastavená veľkosť a pozícia. Ľavému nápisu je nastavená farba na bielu, pravému nápisu je nastavená čierna farba. Ďalej je vytvorený nápis označujúci horné zakrivenie strán. Pod ním je vytvorený posuvník typu `QSlider`.

```
self.top_page_slider.setMaximum(5)
self.top_page_slider.setOrientation(QtCore.Qt.Horizontal)
self.top_page_slider.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
self.top_page_slider.valueChanged.connect(self.top_slider)
```

Posuvníku je nastavený rozsah hodnôt s maximom 5, ďalej jeho orientácia ako horizontálna, taktiež je nastavená zmena kurzoru po prechode myšou cez posuvník a nakoniec je mu pridelená funkcia `top_slider()`, ktorá sa zavolá vždy, keď sa zmení hodnota posuvníka. Funkcia `top_slider` uloží hodnotu posuvníka do premennej top a zmení text nápisu, ktorý sa nachádza vedľa posuvníka na jeho aktuálnu hodnotu:

```
self.top = self.top_page_slider.value()
self.top_curve_number.setText(str(self.top_page_slider.value()))
```

Vedľa posuvníka a nápisu sa vytvorí widget `QLabel`, do ktorého sa nastaví ikona zobrazujúca horné zakrivenie strany. Pod horným posuvníkom sa nachádza nápis označujúci dolné zakrivanie strán typu `QLabel` a pod ním sa opäť nachádza posuvník typu `QSlider`, ktorý funguje rovnako ako horný posuvník, ale pri zmene jeho hodnoty sa volá funkcia `bottom_slider()`, ktorá uloží aktuálnu hodnotu posuvníka do premennej `bottom` a taktiež zmení text nápisu nachádzajúceho sa vedľa posuvníka. Vedľa tohto nápisu sa opäť nachádza widget `QLabel`, v ktorom sa tentokrát nachádza ikona zobrazujúca dolné zakrivenie strán.

bottom_left_frame

Ďalšou časťou je `bottom_left_frame`, nachádzajúci sa vľavo dole. Jeho veľkosť je 460 pixelov na šírku a 140 pixelov na výšku. Nachádza sa v ľavej časti okna `MainWindow` a je umiestnený 540 pixelov zhora. Je mu nastavená farba rgb(230, 182, 94). V jeho vnútri je vytvorený widget typu `QLabel`, v ktorom je vložený nápis označujúci úpravu farieb a widget `kombobox` typu `QComboBox`. Opäť je nastavená zmena kurzoru po prechode myši cez neho. Pomocou metódy addItem sú doň pridané požadované položky. Táto časť obsahuje aj widget `checkbox` typu `QCheckBox`, pomocou ktorého je možné manuálne označiť okraje dokumentov.

right_frame

Táto časť sa nachádza v pravej časti okna, má veľkosť 541 pixelov na šírku a 421 pixelov na výšku. Nachádza sa na pozícií 460 pixelov zľava a 120 pixelov zhora. Má nastavenú bielu farbu pozadia. V tomto widgete je vytvorená inštancia triedy `ListBoxWidget`, ktorá je typu `QListWidget` a nachádza sa v hornej časti súboru `app.py`. Tento objekt je uložený v premennej `files_list`. Pri vytvorení tohto objektu je do premennej `parent` uložený rodičovský prvok, predaný ako argument pri volaní `ListBosWidget()`, v tomto prípade objekt `MainWindow`. Ďalej je vytvorený prázdny zoznam `paths`, do ktorého budú uložené cesty k vstupným súborom. Objektu sú nastavené rozmery a jeho pozícia v okne. Pozícia je nastavená vzhľadom na `MainWindow`, čo je jeho rodičovský objekt a nie na `right_frame` a to z dôvodu, ktorý bude vysvetlený pri mazaní položiek. Pre tento widget je nastavený drag-and-drop, čiže je možné presúvanie súborov do tohto widgetu. Je to možné vďaka funkciám:

```
def dragEnterEvent(self, event):
    if event.mimeData().hasUrls:
        event.accept()
```

```
    else:
        event.ignore()

def dragMoveEvent(self, event):
    if event.mimeData().hasUrls():
        event.setDropAction(Qt.CopyAction)
        event.accept()
    else:
        event.ignore()
```

Ďalšou dôležitou funkciou je funkcia `dropEvent()`. Tá sa stará o spracovanie súborov, ktoré boli do okna pridané štýlom drag-and-drop. V jej tele sa nachádza cyklus, ktorý spracováva všetky položky, ktoré boli pridané do tohto widgetu. Ďalej sa overí, či sa nahraný súbor nachádza v danom zariadení, teda má určenú cestu. Následne je cesta k súboru získaná metódou `toLocalFile()` a uložená do premennej `file`.

```
for url in event.mimeData().urls():
    if url.isLocalFile():
        file = str(url.toLocalFile())
```

Nasleduje obmedzenie súborov. Keďže potrebujeme upravovať fotografie, je nutné obmedziť súbory, ktoré budú vstupovať do programu popísaného v Backendovej časti. Povolené sú len súbory s príponou .png, .jpg a .jpeg. Kontrola prebieha pomocou nástroja regex, kde sa tieto prípony hľadajú na konci cesty k súboru v premennej file.

```
    if (re.search(".*.jpg$", file, re.IGNORECASE)) or
        re.search(".*.jpeg$", file, re.IGNORECASE) or
        re.search(".*.png$", file, re.IGNORECASE):
```

Je vytvorená pomocná premenná `found`, ktorá označuje, či sa tento súbor už vo widgete nachádza. Nasleduje cyklus, ktorý prechádza všetky položky nachádzajúce sa v zozname `paths`. AK sa súbor už nachádza v zozname, premenná `found` sa nastaví na `True`.

```
found = False
    for i in self.paths:
        if (i == file):
            found = True
```

Hneď za cyklom sa nachádza podmienka `if`, ktorá sa riadi premennou `found`. Ak je `found` rovné `False`, do widgetu sa pridá ikona obrázku s názvom konkrétneho súboru pomocou `QListWidgetItem()`.

```
if not found:
    QListWidgetItem(QIcon(file), os.path.basename(file), self)
    self.paths.append(file)
```

Kľudne sa môže stať, že vo widgete budú nachádzať dva súbory s rovnakým názvom, ale budú mať iné umiestnenie. Nemôže sa stať, aby sa vo widgete nachádzal dvakrát ten istý súbor. Po skončení cyklu, ktorý prechádza všetky vložené súbory, sa nachádza podmienka, ktorá kontroluje, či je počet prvkov vo widgete rôzny od nuly. Ak to platí, nastaví sa vzhľad widgetu na hodnotu premennej `QListWidget_full` a prvá položka vo widgete sa nastaví ako aktívna. Týmto sa ukončí funkcia `dropEvent()`. Táto trieda má ešte jednu zaujímavú funkciu, a tou je `keyPressEvent()`, ktorá pri stlačení klávesy `Delete` na klávesnici zavolá metódu `delete_file()` triedy `MainWindow`, o ktorej sa bude písať v ďalšej časti.

```
def keyPressEvent(self, event):
    if event.key() == Qt.Key_Delete:
        self.parent.delete_file()
```

Po vytvorení inštancie triedy `ListBoxWidget` vo funkcii `setupUi()` sa tomuto objektu nastaví základný štýl nachádzajúci sa v premennej `QListWidget_empty`. Tento štýl obsahuje aj nastavenie obrázku na pozadí, ktorý informuje užívateľa o možnosti drag-and-drop. Po nastavení štýlu je objektu priradená funkcia `list_changed()`:

```
self.files_list.currentItemChanged.connect(self.list_changed)
```

Táto funkcia má na starosti nastavenie štýlu tlačidla na spustenie programu a to v prípade, že je nastavený výstupný priečinok. Funkcia zmení štýl tlačidla na hodnotu v premennej `run_button_on`, čo znamená, že tlačidlo zmení farbu. Taktiež sa zmení kurzor po prechode myšou cez toto tlačidlo. To všetko má užívateľovi indikovať, že je možné spustiť program.

Po nastavení widgetu `files_list` sa vytvoria dve tlačidlá: jedno na vkladanie súborov a druhé na ich mazanie. Oba tlačidlá sú typu `QPushButton`. Nastavia sa im veľkosti, pozície, štýly a obom sa pridajú ikony. Ikona pre tlačidlo na vkladanie súborov má znázorňovať vyhľadávanie súborov a tá pre druhé tlačidlo ich mazanie. Taktiež je nastavená zmena kurzoru po prechode myšou cez ne. Pre tlačidlo na vkladanie súborov je pripojená funkcia `import_files()`:

```
self.in_files_button.clicked.connect(self.import_files)
```

Tá funguje podobne ako funkcia `dropEvent()` v triede `ListBoxWidget`, ale je tam pár zásadných rozdielov. Funkcia `dropEvent()` fungovala na princípe drag-and-drop, zatiaľ čo funkcia `import_files` vkladá súbory pomocou dialógu. Najskôr je potrebné tento dialóg vytvoriť a nastaviť, aby sa v ňom zobrazovali len priečinky a existujúce súbory:

```
dialog = QFileDialog()
    dialog.setFileMode(QFileDialog.ExistingFiles)
```

Následne je potrebné tento dialóg otvoriť. Po jeho otvorení si užívateľ môže vybrať súbory, ktoré chce upraviť, tie sa následne uložia do premennej filenames.

```
if dialog.exec_():
    filenames = dialog.selectedFiles()
```

V ďalšom kroku sa táto premenná (zoznam) prechádza a vykonáva sa rovnaká činnosť ako vo funkcii `dropEvent()`. Kontrolujú sa prípony súborov, taktiež sa kontrolujú duplicitné súbory a súbory, ktoré prešli kontrolou sa ukladajú do premennej `paths` v objekte `files_list`. Po tom sa zmení štýl objektu `files_list` na hodnotu v premennej `QListWidget_full`. Keďže sa pri vkladaní súborov pomocou dialógového okna nespustí funkcia `list_changed()`, je potrebné zmeniť štýl tlačidla, ktoré spúšťa program, samostatne. Taktiež aj kurzor pre toto tlačidlo. Pre tlačidlo na odstraňovanie súborov je pripojená funkcia `delete_file()`:

```
self.delete_files_button.clicked.connect(self.delete_file)
```

Táto funkcia na začiatku získa aktuálny prvok v objekte `files_list` a zistí jeho index. Pomocou funkcie `takeItem()` sa prvok odstráni. Taktiež je potrebné odstrániť cestu k súboru, ktorý reprezentoval konkrétny prvok. Najskôr sa overí, že zoznam `paths` v objekte `files_list` nie je prázdny a následne sa cesta k súboru odstráni.

```
idx = self.files_list.row(self.files_list.currentItem())
self.files_list.takeItem(self.files_list.row
    (self.files_list.currentItem()))
if len(self.files_list.paths) > 0:
    self.files_list.paths.pop(idx)
```

Ďalším krokom je overenie, či objekt `files_list` obsahuje nejaké prvky. Ak je prázdny, zmení sa jeho štýl na hodnotu v premennej `QListWidget_empty`. Taktiež sa zmení štýl a kurzor tlačidla na spustenie programu na neaktívne.

bottom_right_frame

Poslednou časťou objektu `MainWindow` je widget `bottom_right_frame`. Ten má rozmery 540 pixelov na šírku a 140 pixelov na výšku a nachádza sa na súradniciach 460 pixelov zľava a 539 pixelov zhora. Má bielu farbu pozadia. Je v ňom vytvorený nápis označujúci výstupný priečinok a pod ním widget typu `QLineEdit`, v ktorom sa bude nachádzať cesta k výstupnému adresáru. Pomocou metódy `setDisabled()` je mu nastavené, že sa nedá upravovať. V pravej časti tohto widgetu sa nachádza tlačidlo `QPushButton`, ktorému je nastavená ikona znázorňujúca priečinok. Po prechode myšou cez toto tlačidlo sa zmení kurzor a po kliknutú naň sa zavolá funkcia `output_folder()`. Táto funkcia vytvorí dialóg typu `QFileDialog`, ktorému je nastavené, že bude zobrazovať len priečinky:

```
dialog = QFileDialog()
dialog.setFileMode(QFileDialog.Directory)
```

Po vybratí priečinka sa cesta k nemu uloží do premennej `directory`. Do widgetu `QLineEdit` vytvoreného v predošlom kroku sa uloží text cesty k danému priečinku. Skontroluje sa, či sa vo widgete `files_list` nachádzajú nejaké položky a ak áno, je zmenený štýl tlačidla na spustenie programu a taktiež kurzor po prechode myši cez neho. Ďalším prvkom je widget typu `QCheckBox`, ktorý slúži na to, že ak je zaškrtnutý, tak sa snímky po spracovaní zobrazia. Posledným prvkom v okne MainWindow je tlačidlo `run_button` typu `QPushButton`. Tlačidlo sa nachádza v pravom dolnom rohu okna. Je mu nastavený štýl pre neaktívne tlačidlo a zmenený kurzor. Po kliknutí na tlačidlo sa zavolá funkcia `run()`. V nej sa najskôr do premennej `working_directory` uloží cesta k priečinku, v ktorom sa program spustil. Nasleduje kontrola kurzoru. V prípade že má kurzor hodnotu `QtCore.Qt.ForbiddenCursor`, je kliknutie ignorované. Tento štýl má tlačidlo v prípade, že sa vo widgete `files_list` nenachádzajú žiadne položky alebo ak nie je zadaný výstupný priečinok. Na spustenie programu musia byť na vstupe nejaké súbory a musí byť nastavený výstupný priečinok, do ktorého sa spracované súbory uložia. Ak je podmienka splnená, do objektu scanner, ktorý je inštanciou triedy Program, sú nahraté premenné `two_pages`, `left_page`, `top`, `bottom`, `colors`, `show_files` a `manual`. Ďalej je tlačidlo zablokované po dobu vykonávania programu a je mu zmenený štýl. Nasleduje cyklus, v ktorom sa nastaví text v tlačidle v tvare a/b, kde a je číslo práve spracovávaného súboru a b je celkový počet súborov. To sa tam nachádza z dôvodu, aby užívateľ vedel, že program pracuje a taktiež mu to dáva akúsi spätnú väzbu, aby vedel, koľko to bude približne trvať. Následne sa zavolá funkcia run() triedy Program. Táto metóda sa volá v bloku `try-except` z dôvodu, ak by nastal nejaký problém vo funkcii `run()` triedy `Program`, tak by program nepadol.

```
for i in range(len(self.files_list.paths)):
    self.run_button.setText(str(i) + "/" + str(len(self.files_list.paths)))
    try:
```

```
            self.scanner.run(self.files_list.paths[i])
        except Exception:
            pass
```

Po skončení cyklu sa zmení aktuálny adresár na pôvodný, ktorý bol uložený v premennej `working_directory`, pretože funkcia `run()` triedy `Program` ho zmenila kvôli ukladaniu výstupných súborov na výstupný priečinok. Následne je vyčistený objekt `files_list` a tiež je vyprázdnená jeho premenná `paths`. Následne je zmenený štýl tlačidla `run_button`, je znova aktivované a tiež je zmenený štýl objektu `files_list` na prázdny widget. Program je pripravený na zmenu nastavení, nahratie nových súborov, prípadnú zmenu výstupného priečinka a opätovné spustenie.

Úplne na konci funkcie `setupUi()` sa zavolá funkcia `retranslateUi()`, ktorá nahrá implicitný text do widgetov typu `QLabel` pri spustení aplikácie.

V triede `MainWindow` sa nachádza ešte jedna funkcia, konkrétne `keyPressEvent()`, ktorá funguje rovnako ako tá v triede `ListBoxWidget`, teda pri stlačení klávesy `Delete` na klávesnici zavolá funkciu `delete_file` a tá vymaže aktuálne označený prvok v objekte `files_list`. Finálnu podobu aplikácie je možné vidieť na snímke 5.7.

Vytvorenie inštalácie

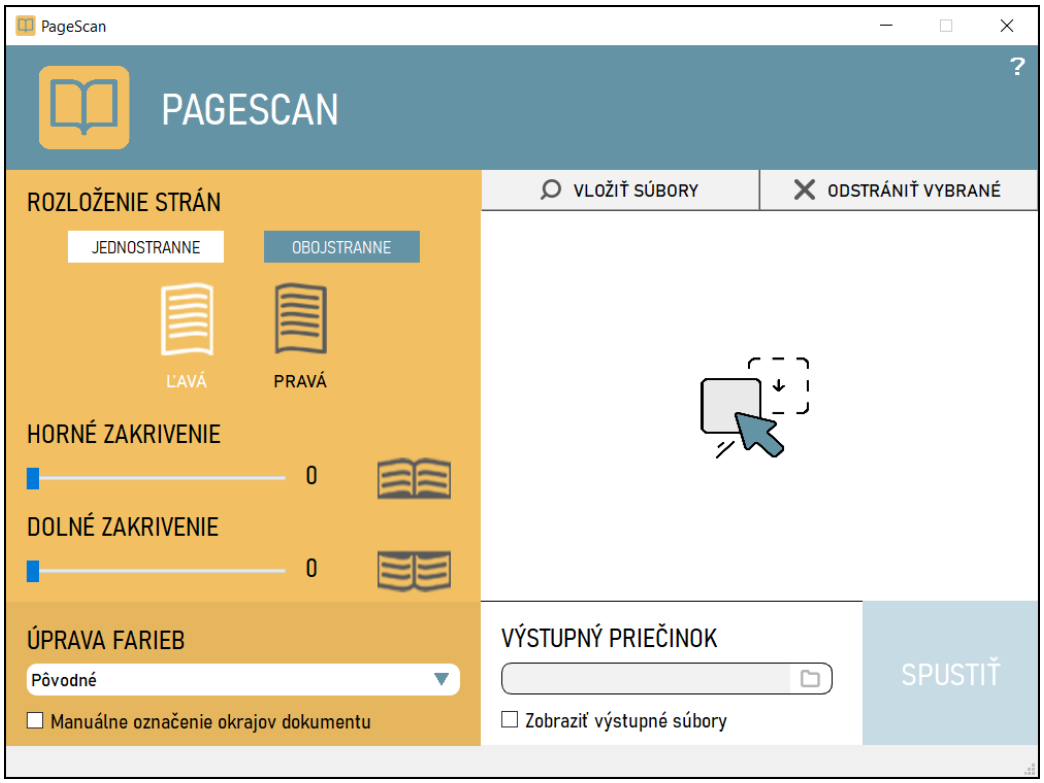
Na to, aby si mohli spustiť tento program aj užívatelia, ktorí nemajú nainštalovaný Python a potrebné knižnice, je potrebné si tento program nainštalovať. Inštalačný súbor bol vytvorený pomocou programu `Inno Setup Compiler`. Aby bolo možné použiť tento program, musí existovať spustiteľný `.exe` súbor vytvoreného programu. Ten bol vytvorený pomocou nástroja `pyinstaller` zadaním príkazu:

```
pyinstaller --onefile -w app.py
```

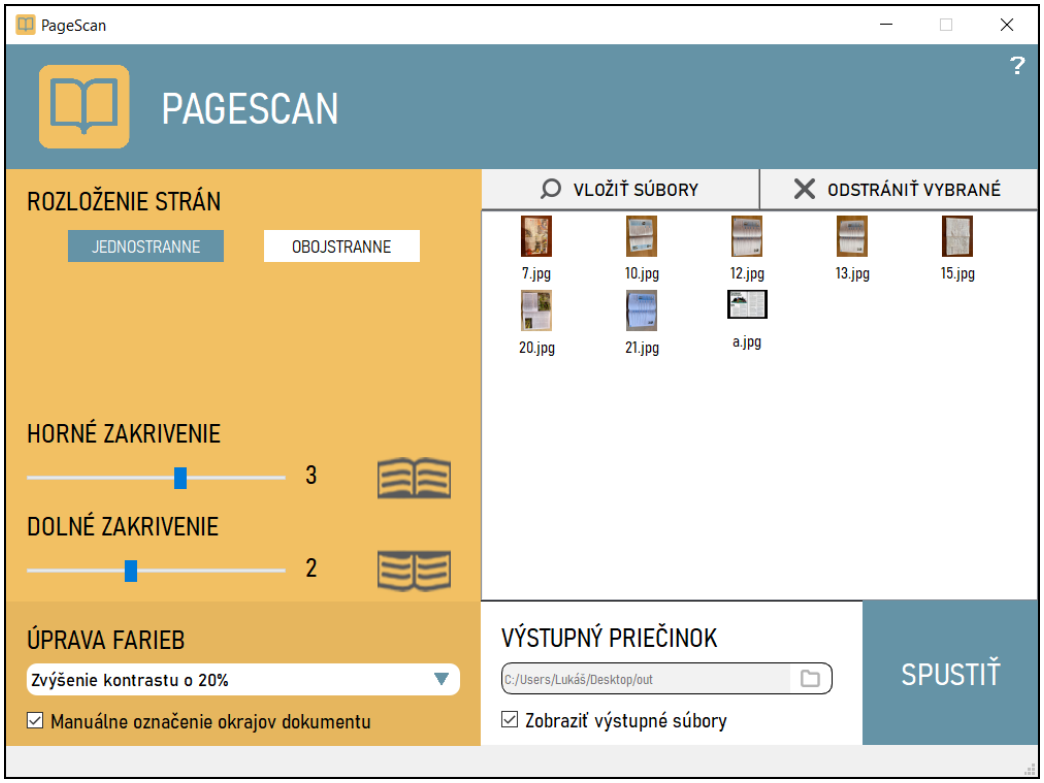
kde:

- –**onefile** zabezpečí to, že sa vytvorí len jeden spustiteľný súbor
- **-w** otvorí len okno programu bez zobrazenia konzoly
- **app.py** je hlavný program

Po vytvorení `.exe` súboru je v programe `Inno Setup Compiler` vytvorený script, ktorý vytvorí inštalačný súbor, z ktorého je možné si nainštalovať aplikáciu SCANNER.



(a) Výsledná aplikácia po spustení.



(b) Výsledná aplikácia po vložení súborov.

Obr. 5.7: **Výsledná aplikácia**

Kapitola 6

Obmedzenia a testovanie

6.1 Obmedzenia

Keďže je program napísaný v jazyku Python, nemal by byť problém s jeho spustením na rôznych platformách, je však potrebné mať nainštalované Python a knižnice v potrebných verziách. Primárne bol však tento program vyvíjaný pre operačný systém Windows 10 a na ňom bol aj testovaný. Obmedzenie sa však týka súborov, s ktorými je možné v programe pracovať. Program dokáže pracovať so súbormi vo formáte .png, .jpg a .jpeg. Toto obmedzenie má program z toho dôvodu, že algoritmy, ktoré sú v ňom použité, nedokážu pracovať s istými formátmi. Napríklad nie je možné pracovať so súbormi vo formáte .pdf, alebo dokonca napríklad .mp3, ak by to nejakého užívateľa napadlo skúsiť. Obmedzenie sa týka aj umiestnenia vstupných súborov. Cesta k nim nemôže obsahovať znaky s diakritikou, pretože sa súbory nepodarí otvoriť.

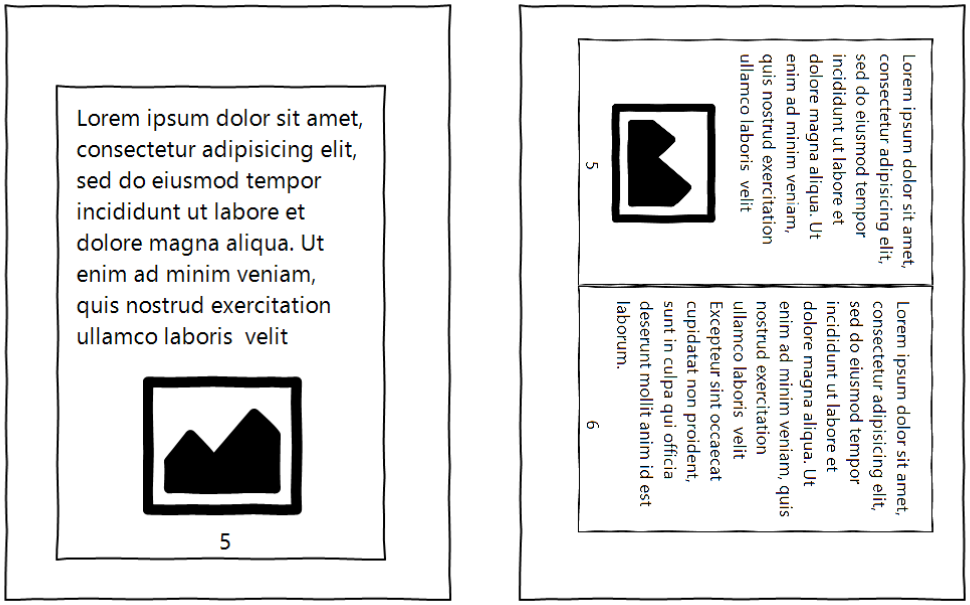
Ďalšia vec nie je obmedzenie, skôr odporúčanie. Ide o nahratie snímok, na ktorých sú odfotené dokumenty v správnom smere a na správnej pozícií. Ideálne by bolo, aby dokument zaberal čo najväčšiu plochu na snímke. Čim väčšiu plochu bude dokument na snímke zaberať, tým lepšia bude jeho výsledná kvalita. To isté platí aj pre rozlíšenie vstupných snímok. Nie je možné očakávať vysokú kvalitu výslednej snímky, keď je pôvodná kvalita veľmi nízka. Ďalšou vecou je minimalizovať zahnutie strán. Aj keď je tento program celkom úspešný vo vyrovnávaní zahnutých strán, najlepšie výsledky budú mať samozrejme snímky, na ktorých sú dokumenty vyrovnané. Dôležité je taktiež to, aby boli strany dokumentov vyfotené kolmo na fotoaparát. To zabezpečí vyššiu výslednú kvalitu, ako keby bol dokument vyfotený pod uhlom, pretože môže dôjsť ku drobným chybám pri aplikovaní perspektívnej transformácie.

Aby nedošlo k nesprávnemu orezaniu a transformácií, dokumenty musia byť na snímke otočené tak, ako ukazuje obrázok 6.1.

6.2 Testovanie

Testovanie prebiehalo na operačnom systéme Windows 10. Testy sa vykonávali na programe spustenom pomocou Pythonu, ale aj na výslednej nainštalovanej aplikácií. Bolo vykonaných niekoľko druhov testov, ktoré boli zamerané na:

- vkladanie súborov,
- jednostranné dokumenty,



Obr. 6.1: **Pozícia dokumentov na snímke**

- dvojstranné dokumenty,
- ohnutie strán,
- na úpravu jasu a farieb.

Testy na vkladanie súborov slúžili na overenie dialógu na vkladanie súborov aj možnosti presúvania súborov spôsobom drag-and-drop. Taktiež bola overená funkčnosť tlačidla na mazanie súborov aj klávesy Delete, po ktorej sa súbory zmazali. Pri týchto testoch bolo zistené, že nie je možné nahrať súbory v prípade, že cesta k nim obsahuje diakritiku z dôvodu obmedzení funkcií knižnice OpenCV.

Testy na jednostranné dokumenty pozostávali z vkladania takýchto dokumentov do programu. Očakávalo sa, že tieto dokumenty program automaticky na snímke odhalí a oreže. To sa podarilo pomerne úspešne. V pár prípadoch bolo potrebné manuálne označiť okraje dokumentu a program pokračoval ďalej. Rovnaké testy boli vykonané aj pre obojstranné dokumenty.

Testy na ohnutie strán boli vykonávané na snímkach otvorených kníh a novín. Boli testované rôzne veľkosti zahnutia strán. Ak boli správne zadané parametre horného a dolného zahnutia, výsledné snímky mali minimalizované zahnutia strán. Menej kvalitné výsledky samozrejme mali snímky, na ktorých boli dokumenty vyfotené pod väčším uhlom a mali aj zahnuté strany.

Testy na úpravu jasu a farieb boli vykonávané na snímkach, kde bolo potrebné zvýšiť, prípadne znížiť kontrast. Taktiež bola otestovaná možnosť konvertovania snímky do čiernobielej (binárnej) podoby. Počas testov boli upravené parametre funkcií, aby bol docielený čo najlepší výsledok.

Kapitola 7

Záver

Cieľom práce bolo vytvoriť knižnicu, ktorá dokáže nájsť okraje dokumentov na snímkach, dokumenty orezať, vyrovnať s rovinou fotoaparátu, použiť metódy na jasové úpravy obrázkov a vyrovnať zahnuté okraje strán. Túto knižnicu bolo potrebné implementovať v programe, ktorý je užívateľsky čo najpríjemnejší.

Cieľ práce bol splnený, bol vytvorený program, ktorý na snímkach dokáže nájsť knihy alebo dokumenty, orezať ich, vyrovnať zahnuté strany a vykonať jasové úpravy podľa potreby. Program sa skladá z dvoch hlavných častí, obe sú napísané v programovacom jazyku Python. Prvou časťou je knižnica, ktorá sa stará o prácu so snímkami, detekciu okrajov strán, ich orezanie, vyrovnanie a úpravy jasu. Využíva na to knižnice OpenCV a Numpy. Druhou časťou je program s grafickým užívateľským rozhraním, ktorý využíva spomínanú knižnicu a taktiež knižnicu PyQt. Knižnica aj program boli dôkladne otestované na základe rôznych scenárov: do programu boli vkladané snímky, ktoré obsahovali rôzne dokumenty, napríklad knihy, časopisy, učebnice, ale aj rôzne dokumenty formátu A4. Boli overené nastavenia, ktorými je v programe možné určiť typ dokumentu, veľkosť zahnutia strán a možnosti úpravy jasu. Výsledky programu sú pomerne dobré. Ak sú zvolené správne nastavenia programu, je možné docieliť kvalitné výsledky. Detekcia strán je celkom úspešná, len občas, ak je medzi dokumentom a podložkou nedostatočný kontrast, je potrebné manuálne označenie okrajov strán. Inak program pracuje správne a plynulo.

V práci je možné pokračovať zdokonalením vytvorenej knižnice, napríklad automatickým odlíšením jednostranných dokumentov od dvojstranných, pretože momentálne sa to musí nastaviť manuálne. Na to je však potrebné ďalšie štúdium v tejto oblasti.

Prínosy tejto práce hodnotím pozitívne, pomohla mi pochopiť problémy spojené so spracovaním obrazu a taktiež zefektívniť vyhľadávanie informácií a ich správne použitie v texte. Taktiež ma tvorba programu motivovala pre ďalšie štúdium v oblasti vývoja softvéru.

Literatúra

[1] Bezryadin, S., Bourov, P. a Ilinih, D. Brightness Calculation in Digital Image Processing. *International Symposium on Technologies for Digital Photo Fulfillment*. 1. vyd. 2007, zv. 2007, č. 1, s. 10–15. DOI: doi:10.2352/ISSN.2169-4672.2007.1.0.10. ISSN 2169-4664. Dostupné z: https://www.ingentaconnect.com/content/ist/tdpf/2007/00002007/00000001/art00005.

[2] Christensson, P. *Image Scaling* [online]. 2019 [cit. 2021-01-15]. Dostupné z: https://techterms.com/definition/image_scaling.

[3] Dalton, W., DeMuro, J. P. a Turner, B. *Best scanning software of 2020: apps to digitize your paper documents* [online]. 2020 [cit. 2020-12-26]. Dostupné z: https://www.techradar.com/best/best-scanning-software.

[4] Distante, A. a Distante, C. *Geometric Transformations*. 1. vyd. Cham: Springer International Publishing, 2020. 149–208 s. ISBN 978-3-030-42374-2. Dostupné z: https://doi.org/10.1007/978-3-030-42374-2_3.

[5] Fiřt, J. a Holota, R. Digitalizace a zpracování obrazu. [online]. 2008, [cit. 2020-12-27]. Dostupné z: http://home.zcu.cz/~holota5/publ/DigZpr0.pdf.

[6] Igor, F. *Zpracování obrazu v embedded systémech*. Brno, CZ, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedúci práce Najman, I. P.

[7] Keough, B. *The Best Mobile Scanning Apps* [online]. 2020 [cit. 2020-12-26]. Dostupné z: https://www.nytimes.com/wirecutter/reviews/best-mobile-scanning-apps/.

[8] Muthukrishnan, R. a Radha, M. EDGE DETECTION TECHNIQUES FOR IMAGE SEGMENTATION. *International Journal of Computer Science & Information Technology*. December 2011, zv. 03.

[9] Mutneja, V. Methods of Image Edge Detection: A Review. *Journal of Electrical & Electronic Systems*. Január 2015, zv. 04.

[10] Nadernejad, E., Sharifzadeh, S. a Hassanpour, H. Edge Detection Techniques: Evaluations and Comparison. *Applied Mathematical Sciences*. Hikari Ltd. 2008, zv. 2, č. 31, s. 1507–1520. ISSN 1312-885X.

[11] Přemysl, K. *Základy počítačové grafiky* [online]. 2005 [cit. 2020-12-28]. Dostupné z: https://wis.fit.vutbr.cz/FIT/st/cfs.php.cs?file=%2Fcourse%2FIZG-IT%2Ftexts%2Fizg_opora.pdf&cid=13375.

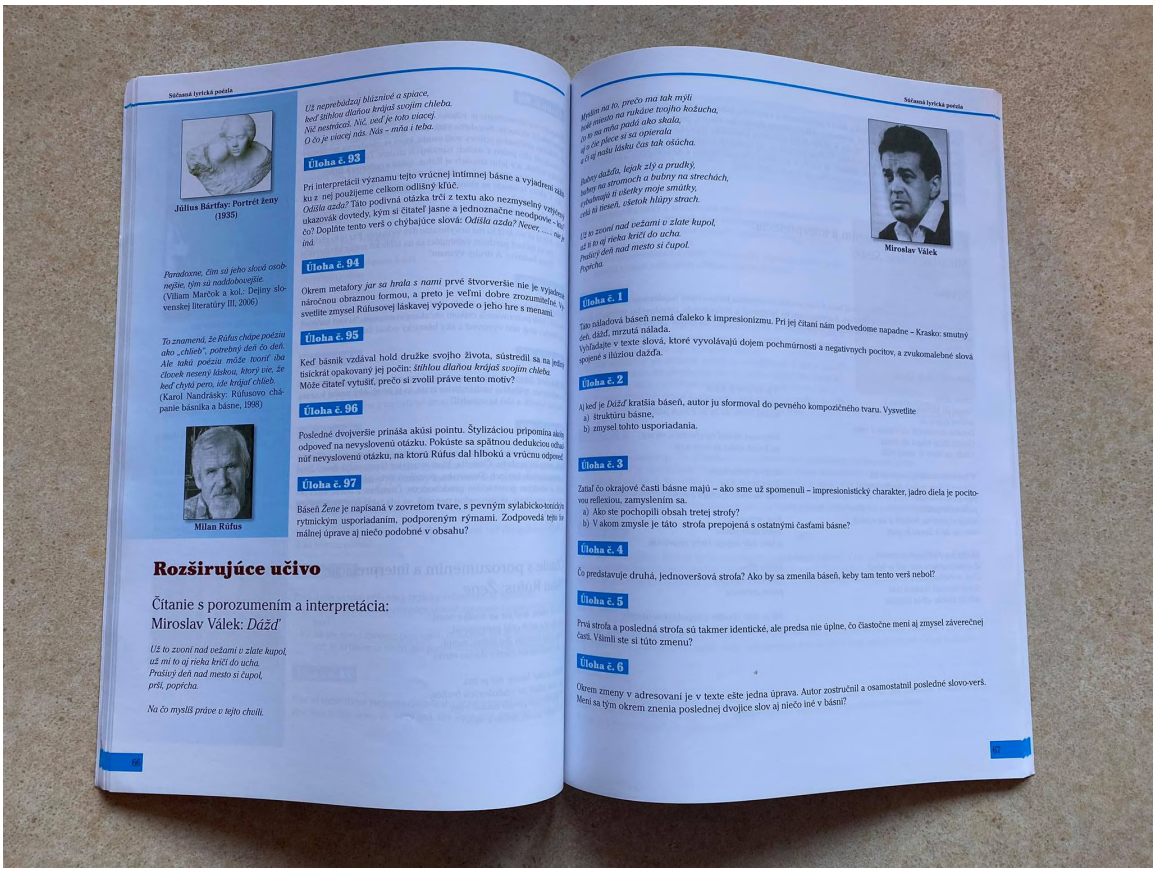
[12] R. Fisher, A. W. a Wolfart., E. *Gaussian Smoothing* [online]. 2003 [cit. 2021-01-17]. Dostupné z: https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm.

[13] Wikipedia contributors. *Comparison gallery of image scaling algorithms — Wikipedia, The Free Encyclopedia* [https://en.wikipedia.org/w/index.php?title=Comparison_gallery_of_image_scaling_algorithms&oldid=989411683]. 2020 [cit. 2021-01-16].

[14] Wikipedia contributors. *Gaussian blur — Wikipedia, The Free Encyclopedia* [https://en.wikipedia.org/w/index.php?title=Gaussian_blur&oldid=1000984962]. 2021 [cit. 2021-01-17].

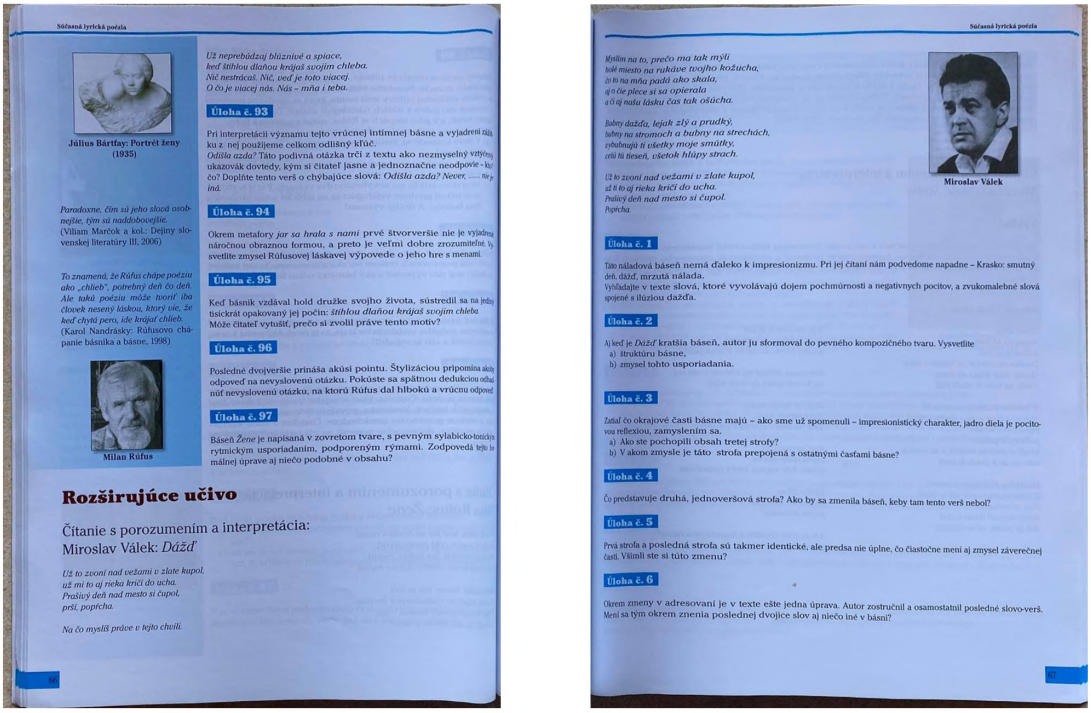
Príloha A

Výsledky programu

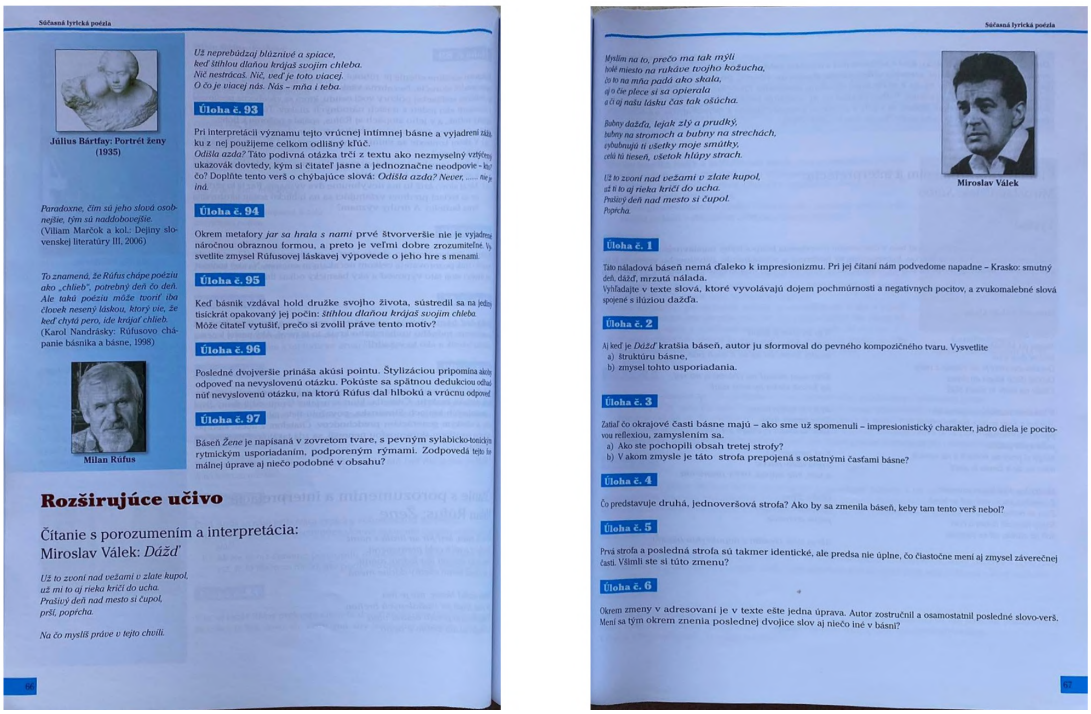


Obr. A.1: **Vyfotený obrázok**

Automatický režim, horné zakrivenie: 3, dolné zakrivenie: 2

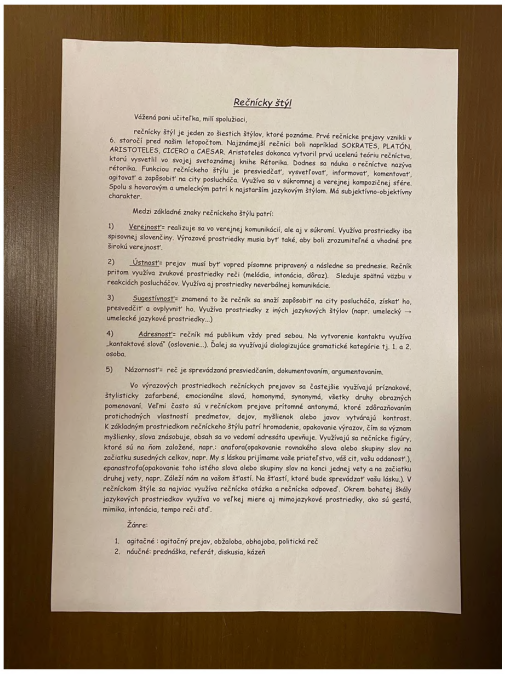


Manuálny režim, horné zakrivenie: 3, dolné zakrivenie: 2



Obr. A.2: **Dokumenty na snímke detekované automaticky a manuálne s využitím korekcie zakrivených okrajov strán**

Oroginálny obrázok



Rečnícky štýl

Vážená pani učiteľka, milí spolužiaci,

rečnícky štýl je jeden zo šiestich štýlov, ktoré poznáme. Prvé rečnícke prejavy vznikli v 6. storočí pred našim letopočtom. Najznámejší rečníci boli napríklad SOKRATES, PLATÓN, ARISTOTELES, CICERO a CAESAR. Aristoteles dokonca vytvoril prvú ucelenú teóriu rečníctva, ktorú vysvetlil vo svojej svetoznámej knihe Rétorika. Dodnes sa náuka o rečníctve nazýva rétorika. Funkciou rečníckeho štýlu je presvedčať, vysvetľovať, informovať, komentovať, agitovať a zapôsobiť na city poslucháča. Využíva sa v súkromnej a verejnej kompozičnej sfére. Spolu s hovorovým a umeleckým patrí k najstarším jazykovým štýlom. Má subjektívno-objektívny charakter.

Medzi základné znaky rečníckeho štýlu patrí:

1) Verejnosť= realizuje sa vo verejnej komunikácii, ale aj v súkromí. Využíva prostriedky iba spisovnej slovenčiny. Výrazové prostriedky musia byť také, aby boli zrozumiteľné a vhodné pre širokú verejnosť.

2) Ústnosť= prejav musí byť vopred písomne pripravený a následne sa prednesie. Rečník pritom využíva zvukové prostriedky reči (melódia, intonácia, dôraz). Sleduje spätnú väzbu v reakciách poslucháčov. Využíva aj prostriedky neverbálnej komunikácie.

3) Sugestívnosť= znamená to že rečník sa snaží zapôsobiť na city poslucháča, získať ho, presvedčiť a ovplyvniť ho. Využíva prostriedky z iných jazykových štýlov (napr. umelecký → umelecké jazykové prostriedky...)

4) Adresnosť= rečník má publikum vždy pred sebou. Na vytvorenie kontaktu využíva „kontaktové slová" (oslovenie...). Ďalej sa využívajú dialogizujúce gramatické kategórie tj. 1. a 2. osoba.

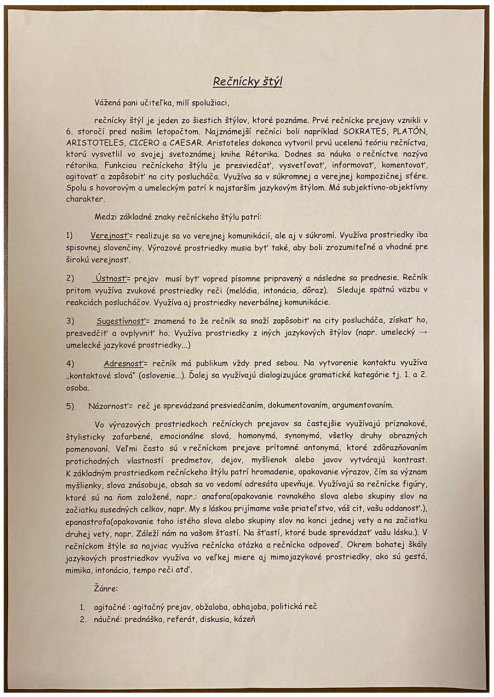
5) Názornosť= reč je sprevádzaná presviedčaním, dokumentovaním, argumentovaním.

Vo výrazových prostriedkoch rečníckych prejavov sa častejšie využívajú príznakové, štylisticky zafarbené, emocionálne slová, homonymá, synonymá, všetky druhy obrazných pomenovaní. Veľmi často sú v rečníckom prejave prítomné antonymá, ktoré zdôrazňovaním protichodných vlastností predmetov, dejov, myšlienok alebo javov vytvárajú kontrast. K základným prostriedkom rečníckeho štýlu patrí hromadenie, opakovanie výrazov, čím sa význam myšlienky, slova znásobuje, obsah sa vo vedomí adresáta upevňuje. Využívajú sa rečnícke figúry, ktoré sú na ňom založené, napr.: anafora(opakovanie rovnakého slova alebo skupiny slov na začiatku susedných celkov, napr. My s láskou prijímame vaše priateľstvo, váš cit, vašu oddanosť.), epanastrofa(opakovanie toho istého slova alebo skupiny slov na konci jednej vety a na začiatku druhej vety, napr. Záleží nám na vašom šťastí. Na šťastí, ktoré bude sprevádzať vašu lásku.). V rečníckom štýle sa najviac využíva rečnícka otázka a rečnícka odpoveď. Okrem bohatej škály jazykových prostriedkov využíva vo veľkej miere aj mimojazykové prostriedky, ako sú gestá, mimika, intonácia, tempo reči atď.

Žánre:

1. agitačné : agitačný prejav, obžaloba, obhajoba, politická reč
2. náučné: prednáška, referát, diskusia, kázeň

Automaticky orezaný obrázok, bez úpravy farieb



Rečnícky štýl

Vážená pani učiteľka, milí spolužiaci,

rečnícky štýl je jeden zo šiestich štýlov, ktoré poznáme. Prvé rečnícke prejavy vznikli v 6. storočí pred našim letopočtom. Najznámejší rečníci boli napríklad SOKRATES, PLATÓN, ARISTOTELES, CICERO a CAESAR. Aristoteles dokonca vytvoril prvú ucelenú teóriu rečníctva, ktorú vysvetlil vo svojej svetoznámej knihe Rétorika. Dodnes sa náuka o rečníctve nazýva rétorika. Funkciou rečníckeho štýlu je presvedčať, vysvetľovať, informovať, komentovať, agitovať a zapôsobiť na city poslucháča. Využíva sa v súkromnej a verejnej kompozičnej sfére. Spolu s hovorovým a umeleckým patrí k najstarším jazykovým štýlom. Má subjektívno-objektívny charakter.

Medzi základné znaky rečníckeho štýlu patrí:

1) Verejnosť= realizuje sa vo verejnej komunikácii, ale aj v súkromí. Využíva prostriedky iba spisovnej slovenčiny. Výrazové prostriedky musia byť také, aby boli zrozumiteľné a vhodné pre širokú verejnosť.

2) Ústnosť= prejav musí byť vopred písomne pripravený a následne sa prednesie. Rečník pritom využíva zvukové prostriedky reči (melódia, intonácia, dôraz). Sleduje spätnú väzbu v reakciách poslucháčov. Využíva aj prostriedky neverbálnej komunikácie.

3) Sugestívnosť= znamená to že rečník sa snaží zapôsobiť na city poslucháča, získať ho, presvedčiť a ovplyvniť ho. Využíva prostriedky z iných jazykových štýlov (napr. umelecký → umelecké jazykové prostriedky...)

4) Adresnosť= rečník má publikum vždy pred sebou. Na vytvorenie kontaktu využíva „kontaktové slová" (oslovenie...). Ďalej sa využívajú dialogizujúce gramatické kategórie tj. 1. a 2. osoba.

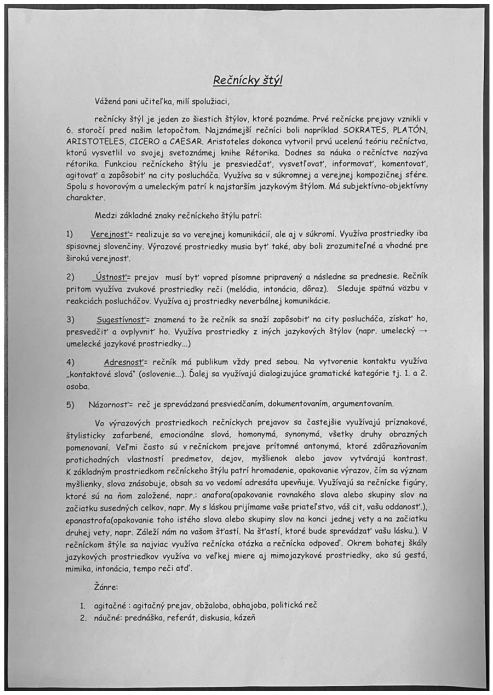
5) Názornosť= reč je sprevádzaná presviedčaním, dokumentovaním, argumentovaním.

Vo výrazových prostriedkoch rečníckych prejavov sa častejšie využívajú príznakové, štylisticky zafarbené, emocionálne slová, homonymá, synonymá, všetky druhy obrazných pomenovaní. Veľmi často sú v rečníckom prejave prítomné antonymá, ktoré zdôrazňovaním protichodných vlastností predmetov, dejov, myšlienok alebo javov vytvárajú kontrast. K základným prostriedkom rečníckeho štýlu patrí hromadenie, opakovanie výrazov, čím sa význam myšlienky, slova znásobuje, obsah sa vo vedomí adresáta upevňuje. Využívajú sa rečnícke figúry, ktoré sú na ňom založené, napr.: anafora(opakovanie rovnakého slova alebo skupiny slov na začiatku susedných celkov, napr. My s láskou prijímame vaše priateľstvo, váš cit, vašu oddanosť.), epanastrofa(opakovanie toho istého slova alebo skupiny slov na konci jednej vety a na začiatku druhej vety, napr. Záleží nám na vašom šťastí. Na šťastí, ktoré bude sprevádzať vašu lásku.). V rečníckom štýle sa najviac využíva rečnícka otázka a rečnícka odpoveď. Okrem bohatej škály jazykových prostriedkov využíva vo veľkej miere aj mimojazykové prostriedky, ako sú gestá, mimika, intonácia, tempo reči atď.

Žánre:

1. agitačné : agitačný prejav, obžaloba, obhajoba, politická reč
2. náučné: prednáška, referát, diskusia, kázeň

Obrázok v odtieňoch šedej



Rečnícky štýl

Vážená pani učiteľka, milí spolužiaci,

rečnícky štýl je jeden zo šiestich štýlov, ktoré poznáme. Prvé rečnícke prejavy vznikli v 6. storočí pred našim letopočtom. Najznámejší rečníci boli napríklad SOKRATES, PLATÓN, ARISTOTELES, CICERO a CAESAR. Aristoteles dokonca vytvoril prvú ucelenú teóriu rečníctva, ktorú vysvetlil vo svojej svetoznámej knihe Rétorika. Dodnes sa náuka o rečníctve nazýva rétorika. Funkciou rečníckeho štýlu je presvedčať, vysvetľovať, informovať, komentovať, agitovať a zapôsobiť na city poslucháča. Využíva sa v súkromnej a verejnej kompozičnej sfére. Spolu s hovorovým a umeleckým patrí k najstarším jazykovým štýlom. Má subjektívno-objektívny charakter.

Medzi základné znaky rečníckeho štýlu patrí:

1) Verejnosť= realizuje sa vo verejnej komunikácii, ale aj v súkromí. Využíva prostriedky iba spisovnej slovenčiny. Výrazové prostriedky musia byť také, aby boli zrozumiteľné a vhodné pre širokú verejnosť.

2) Ústnosť= prejav musí byť vopred písomne pripravený a následne sa prednesie. Rečník pritom využíva zvukové prostriedky reči (melódia, intonácia, dôraz). Sleduje spätnú väzbu v reakciách poslucháčov. Využíva aj prostriedky neverbálnej komunikácie.

3) Sugestívnosť= znamená to že rečník sa snaží zapôsobiť na city poslucháča, získať ho, presvedčiť a ovplyvniť ho. Využíva prostriedky z iných jazykových štýlov (napr. umelecký → umelecké jazykové prostriedky...)

4) Adresnosť= rečník má publikum vždy pred sebou. Na vytvorenie kontaktu využíva „kontaktové slová" (oslovenie...). Ďalej sa využívajú dialogizujúce gramatické kategórie tj. 1. a 2. osoba.

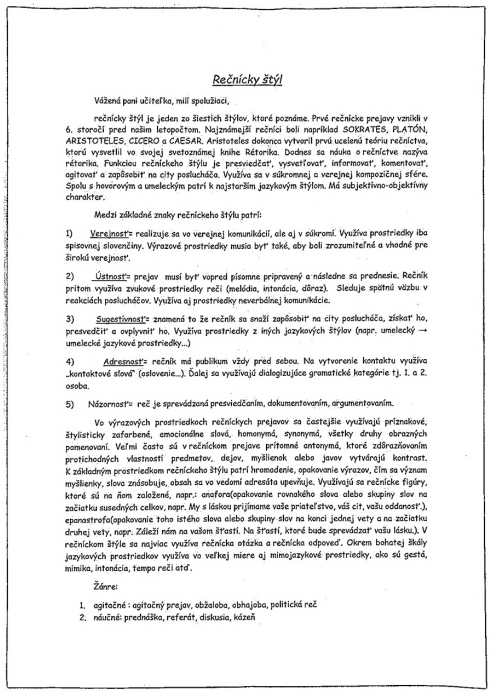
5) Názornosť= reč je sprevádzaná presviedčaním, dokumentovaním, argumentovaním.

Vo výrazových prostriedkoch rečníckych prejavov sa častejšie využívajú príznakové, štylisticky zafarbené, emocionálne slová, homonymá, synonymá, všetky druhy obrazných pomenovaní. Veľmi často sú v rečníckom prejave prítomné antonymá, ktoré zdôrazňovaním protichodných vlastností predmetov, dejov, myšlienok alebo javov vytvárajú kontrast. K základným prostriedkom rečníckeho štýlu patrí hromadenie, opakovanie výrazov, čím sa význam myšlienky, slova znásobuje, obsah sa vo vedomí adresáta upevňuje. Využívajú sa rečnícke figúry, ktoré sú na ňom založené, napr.: anafora(opakovanie rovnakého slova alebo skupiny slov na začiatku susedných celkov, napr. My s láskou prijímame vaše priateľstvo, váš cit, vašu oddanosť.), epanastrofa(opakovanie toho istého slova alebo skupiny slov na konci jednej vety a na začiatku druhej vety, napr. Záleží nám na vašom šťastí. Na šťastí, ktoré bude sprevádzať vašu lásku.). V rečníckom štýle sa najviac využíva rečnícka otázka a rečnícka odpoveď. Okrem bohatej škály jazykových prostriedkov využíva vo veľkej miere aj mimojazykové prostriedky, ako sú gestá, mimika, intonácia, tempo reči atď.

Žánre:

1. agitačné : agitačný prejav, obžaloba, obhajoba, politická reč
2. náučné: prednáška, referát, diskusia, kázeň

Čierno-biely obrázok



Rečnícky štýl

Vážená pani učiteľka, milí spolužiaci,

rečnícky štýl je jeden zo šiestich štýlov, ktoré poznáme. Prvé rečnícke prejavy vznikli v 6. storočí pred našim letopočtom. Najznámejší rečníci boli napríklad SOKRATES, PLATÓN, ARISTOTELES, CICERO a CAESAR. Aristoteles dokonca vytvoril prvú ucelenú teóriu rečníctva, ktorú vysvetlil vo svojej svetoznámej knihe Rétorika. Dodnes sa náuka o rečníctve nazýva rétorika. Funkciou rečníckeho štýlu je presvedčať, vysvetľovať, informovať, komentovať, agitovať a zapôsobiť na city poslucháča. Využíva sa v súkromnej a verejnej kompozičnej sfére. Spolu s hovorovým a umeleckým patrí k najstarším jazykovým štýlom. Má subjektívno-objektívny charakter.

Medzi základné znaky rečníckeho štýlu patrí:

1) Verejnosť= realizuje sa vo verejnej komunikácii, ale aj v súkromí. Využíva prostriedky iba spisovnej slovenčiny. Výrazové prostriedky musia byť také, aby boli zrozumiteľné a vhodné pre širokú verejnosť.

2) Ústnosť= prejav musí byť vopred písomne pripravený a následne sa prednesie. Rečník pritom využíva zvukové prostriedky reči (melódia, intonácia, dôraz). Sleduje spätnú väzbu v reakciách poslucháčov. Využíva aj prostriedky neverbálnej komunikácie.

3) Sugestívnosť= znamená to že rečník sa snaží zapôsobiť na city poslucháča, získať ho, presvedčiť a ovplyvniť ho. Využíva prostriedky z iných jazykových štýlov (napr. umelecký → umelecké jazykové prostriedky...)

4) Adresnosť= rečník má publikum vždy pred sebou. Na vytvorenie kontaktu využíva „kontaktové slová" (oslovenie...). Ďalej sa využívajú dialogizujúce gramatické kategórie tj. 1. a 2. osoba.

5) Názornosť= reč je sprevádzaná presviedčaním, dokumentovaním, argumentovaním.

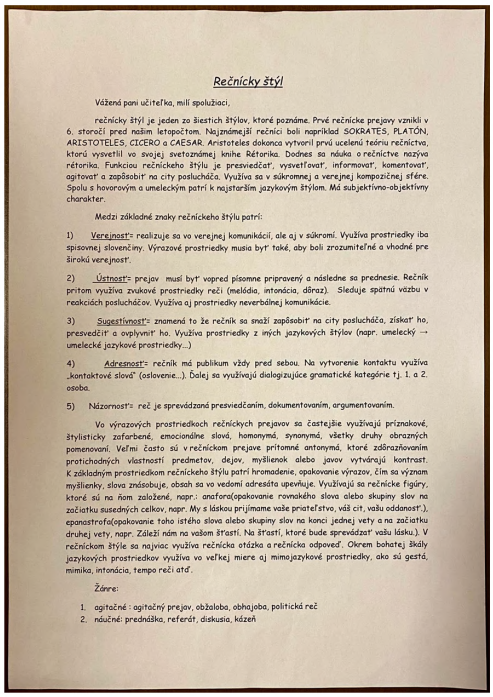
Vo výrazových prostriedkoch rečníckych prejavov sa častejšie využívajú príznakové, štylisticky zafarbené, emocionálne slová, homonymá, synonymá, všetky druhy obrazných pomenovaní. Veľmi často sú v rečníckom prejave prítomné antonymá, ktoré zdôrazňovaním protichodných vlastností predmetov, dejov, myšlienok alebo javov vytvárajú kontrast. K základným prostriedkom rečníckeho štýlu patrí hromadenie, opakovanie výrazov, čím sa význam myšlienky, slova znásobuje, obsah sa vo vedomí adresáta upevňuje. Využívajú sa rečnícke figúry, ktoré sú na ňom založené, napr.: anafora(opakovanie rovnakého slova alebo skupiny slov na začiatku susedných celkov, napr. My s láskou prijímame vaše priateľstvo, váš cit, vašu oddanosť.), epanastrofa(opakovanie toho istého slova alebo skupiny slov na konci jednej vety a na začiatku druhej vety, napr. Záleží nám na vašom šťastí. Na šťastí, ktoré bude sprevádzať vašu lásku.). V rečníckom štýle sa najviac využíva rečnícka otázka a rečnícka odpoveď. Okrem bohatej škály jazykových prostriedkov využíva vo veľkej miere aj mimojazykové prostriedky, ako sú gestá, mimika, intonácia, tempo reči atď.

Žánre:

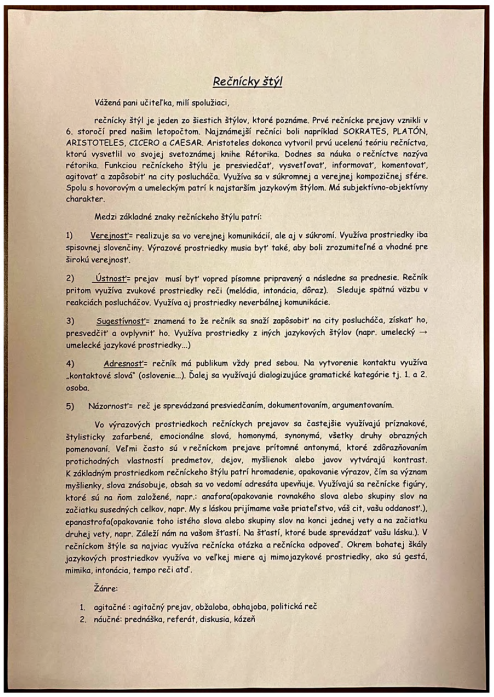
1. agitačné : agitačný prejav, obžaloba, obhajoba, politická reč
2. náučné: prednáška, referát, diskusia, kázeň

Obr. A.3: **Využitie metód na úpravu farieb**

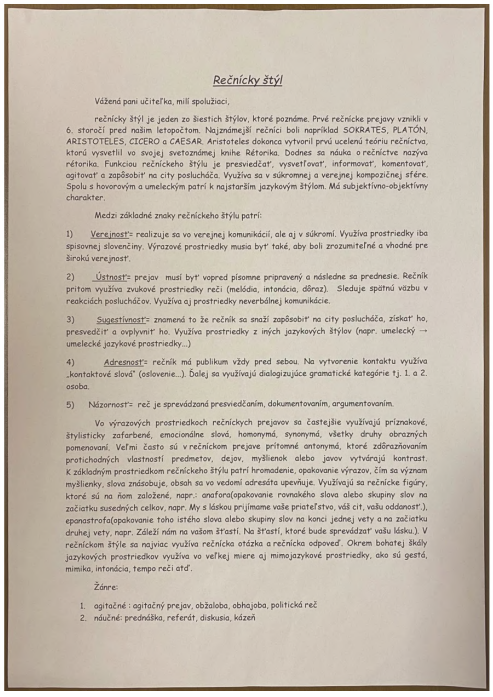
Jas + 20%



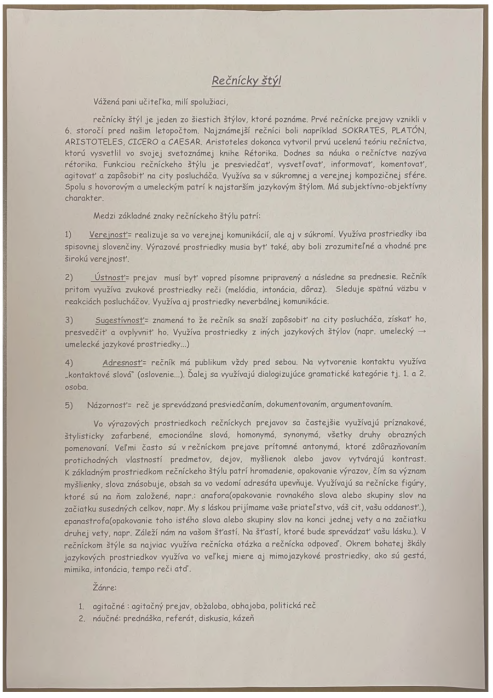
Jas + 40%



Jas - 20%



Jas - 40%



Rečnícky štýl

Vážená pani učiteľka, milí spolužiaci,

rečnícky štýl je jeden zo šiestich štýlov, ktoré poznáme. Prvé rečnícke prejavy vznikli v 6. storočí pred naším letopočtom. Najznámejší rečníci boli napríklad SOKRATES, PLATÓN, ARISTOTELES, CICERO a CAESAR. Aristoteles dokonca vytvoril prvú ucelenú teóriu rečníctva, ktorú vysvetlil vo svojej svetoznámej knihe Rétorika. Dodnes sa náuka o rečníctve nazýva rétorika. Funkciou rečníckeho štýlu je presvedčať, vysvetľovať, informovať, komentovať, agitovať a zapôsobiť na city poslucháča. Využíva sa v súkromnej a verejnej kompozičnej sfére. Spolu s hovorovým a umeleckým patrí k najstarším jazykovým štýlom. Má subjektívno-objektívny charakter.

Medzi základné znaky rečníckeho štýlu patrí:

1) Verejnosť= realizuje sa vo verejnej komunikácii, ale aj v súkromí. Využíva prostriedky iba spisovnej slovenčiny. Výrazové prostriedky musia byť také, aby boli zrozumiteľné a vhodné pre širokú verejnosť.

2) Ústnosť= prejav musí byť vopred písomne pripravený a následne sa prednesie. Rečník pritom využíva zvukové prostriedky reči (melódia, intonácia, dôraz). Sleduje spätnú väzbu v reakciách poslucháčov. Využíva aj prostriedky neverbálnej komunikácie.

3) Sugestívnosť= znamená to že rečník sa snaží zapôsobiť na city poslucháča, získať ho, presvedčiť a ovplyvniť ho. Využíva prostriedky z iných jazykových štýlov (napr. umelecký → umelecké jazykové prostriedky...)

4) Adresnosť= rečník má publikum vždy pred sebou. Na vytvorenie kontaktu využíva „kontaktové slová" (oslovenie...). Ďalej sa využívajú dialogizujúce gramatické kategórie tj. 1. a 2. osoba.

5) Názornosť= reč je sprevádzaná presvedčaním, dokumentovaním, argumentovaním.

Vo výrazových prostriedkoch rečníckych prejavov sa častejšie využívajú príznakové, štylisticky zafarbené, emocionálne slová, homonymá, synonymá, všetky druhy obrazných pomenovaní. Veľmi často sú v rečníckom prejave prítomné antonymá, ktoré zdôrazňovaním protichodných vlastností predmetov, dejov, myšlienok alebo javov vytvárajú kontrast. K základným prostriedkom rečníckeho štýlu patrí hromadenie, opakovanie výrazov, čím sa význam myšlienky, slova znásobuje, obsah sa vo vedomí adresáta upevňuje. Využívajú sa rečnícke figúry, ktoré sú na ňom založené, napr.: anafora(opakovanie rovnakého slova alebo skupiny slov na začiatku susedných celkov, napr. My s láskou prijímame vaše priateľstvo, váš cit, vašu oddanosť.), epanastrofa(opakovanie toho istého slova alebo skupiny slov na konci jednej vety a na začiatku druhej vety, napr. Záleží nám na vašom šťastí. Na šťastí, ktoré bude sprevádzať vašu lásku.). V rečníckom štýle sa najviac využíva rečnícka otázka a rečnícka odpoveď. Okrem bohatej škály jazykových prostriedkov využíva vo veľkej miere aj mimojazykové prostriedky, ako sú gestá, mimika, intonácia, tempo reči atď.

Žánre:

1. agitačné : agitačný prejav, obžaloba, obhajoba, politická reč
2. náučné: prednáška, referát, diskusia, kázeň

Obr. A.4: **Využitie metód na úpravu jasu**

Príloha B

CD

Obsah CD:

- **exe_subor**: priečinok obsahuje spustiteľný program,
- **instalacia**: priečinok obsahuje súbor na inštaláciu programu,
- **technicka_sprava**: priečinok obsahuje zdrojové súbory technickej správy,
- **zdrojove_subory**: priečinok obsahuje všetky zdrojové súbory programu,
- **README.txt**: súbor obsahuje návod na inštaláciu, spustenie a prácu s programom,
- **technicka_sprava.pdf**: PDF súbor technickej správy,
- **technicka_sprava_tlac.pdf**: PDF súbor technickej správy určenej na tlač.